19th

$15.00

ARRL and TAPR DIGITAL COMMUNICATIONS CONFERENCE

ORLANDO, FLORIDA

September 22-24, 2000

19th ARRL and TAPR DIGITAL COMMUNICATIONS CONFERENCE

Co-Hosts:

Packet Radio Users Group of Japan
Lake Monroe Amateur Radio Society
Orange County ARES/RACES
Seminole County ARES/RACES
Orlando Amateur Radio Club

ARRL
225 Main Street
Newington, CT 06111-1494 USA
tel: 860-594-0200 www.arrl.org

Tucson Amateur Packet Radio
8987-309 E Tanque Verde Rd #337
Tucson, AZ 85749-9399 USA
tel: 940-383-0000 www.tapr.org

Printed in USA

ISBN: 0-87259-814-4

ARRL Order Number: 8144

First Edition

Welcome!

The 19th ARRL and TAPR Digital Communication Conference is a living testimonial to the spirit of experimentation and exploration that is at the core of Amateur Radio. Although technological exploration can take place on many fronts, it is obvious that the digital community is spearheading the primary drive into the future.

Just browse these proceedings and you'll notice the incredible variety of creative ideas and projects. The Automatic Position Reporting System (APRS) shows its strength with no less than six articles. And even as PSK31 is sweeping through the HF digital community, new HF modes are following close behind, as you'll read in the article by Murray Greenman, ZL1BPU, and Nino Porcino, IZ8BLY. When it comes to project hardware and firmware, PICs have become extraordinarily popular. Take a look at the "EasyTrak" rotator/radio controller interface by Steven R. Bible, N7HPR, or the PIC-et Radio II by John Hansen, W2FS.

I hope these proceedings and this conference will inspire you to begin some digital explorations of your own—and that we'll have the pleasure of seeing one of your articles in the 2001 edition.

David Sumner, K1ZZ
ARRL Executive Vice President

September 2000

Table of Contents

EasyTrak, A PIC Based Rotor/Radio Controller Interface

by Steven R. Bible, N7HPR (n7hpr@tapr.org)

Abstract

EasyTrak is a rotor/radio controller interface based on the Microchip[1] PIC16F87x series PICmicro® microcontroller. The goal was to design a rotor/radio interface that is compact, low-cost and easy to use. The PIC microcontroller contains many of the peripherals needed to design a rotor/radio controller interface: analog to digital converter (ADC), timers, serial interface (USART), and individually programmable I/O pins. In addition, the PIC contains FLASH ROM for easy programming and upgrades, RAM for system variables, and EEPROM for storage of non-volatile configuration information. With a small amount of interface circuitry around the PIC microcontroller, the goals of compact and low-cost were obtained. Ease of use was obtained by designing EasyTrak to interface to the most popular azimuth/elevation rotors and a serial RS-232 interface to communicate with virtually any computer.

EasyTrak is designed to easily interface to the Yaesu series of azimuth/elevation rotors, model numbers G5400B, G5600, and the newest G5500. These rotors have a computer interface built in with an 8-pin DIN connector on the back of the rotor controller to control left, right, down, up and provide rotor position via a proportional 0-5 VDC analog signal for azimuth and elevation. EasyTrak can be interface to other rotors, for example rotating HF antennas, but will require user modification of the rotor controller. Optional relays can be installed on EasyTrak to provide normally-open (NO) or normally-closed (NC) contacts for azimuth, elevation, and brake.

The computer interface is via a serial RS-232 connection. The rotor controller protocol is based on Chris Jackson's, G7UPN, EasyComm protocol. EasyComm is a simple ASCII character based protocol for controlling rotors and radios. The benefit was that a new protocol did not have to be created, it is easy for programmers to write for and interface to, and several programs already have EasyComm programmed in: WiSP, Nova, MacDoppler, MacAPRS and WinAPRS. The Mac and WinAPRS programs have the unique feature of tracking high and low altitude balloons in azimuth and elevation as well as other objects[2]. Mac and WinAPRS can also point HF beams using the DX Cluster feature or point to a moving object in azimuth only.

Introduction

This project started out several years ago with the goal of designing a low-cost rotor/radio controller interface. EasyTrak was designed primarily with satellite operation in mind. It had to interface easily to the most popular azimuth/elevation rotor system, the Yaesu G5400B/G5600B and G5500 series. The Yaesu rotor controllers have a built in computer interface which supplies a proportional 0-5 V analog signal that corresponds to the rotor position (azimuth and elevation) and four command lines left, right, down, and up (when grounded, command the rotors in the respective direction). EasyTrak also had to control the most popular satellite radios. However, EasyTrak is not limited to satellite operations only. It can easily be configured and interfaced to other rotor controllers but requires the user to modify the rotor controller.

Central to the EasyTrak design is the Microchip PICmicro.® The PIC was a natural choice with the plethora of integrated peripherals in a single integrated circuit package. Integrated peripherals reduce the amount of external circuitry required and help keep costs down. The PICmicro chosen for the EasyTrak project was the PIC16F87x series. The F87x series has a maximum of 8 K FLASH ROM, 368 bytes of RAM, and 256 bytes of EEPROM. The flash ROM provides an easy method to program and upgrade the firmware (verses EPROM) and the EEPROM a means to store configuration values. Onboard peripherals of interest for designing a rotor controller include an eight channel 10-bit Analog to Digital Controller (ADC), one 16-bit and two 8-bit timers, and Universal Synchronous/Asynchronous Receiver Transmitter (USART) for serial communications to a host computer. The PICmicro is the center of the hardware design, but a protocol was needed to control the rotors and radios.

It was important not to create another rotor controller protocol. There already exist several rotor controllers (commercial, kit, and homebrew), each with its own unique protocol. This complicates the software authors' job in keeping up with multiple protocols. What was needed was a protocol that was simple and provided the necessary basic commands.

EasyComm Protocol

The protocol chosen was Chris Jackson's, G7UPN, EasyComm[3] (thus the origin of selecting the name EasyTrak). Chris is the author of the popular digital satellite program WiSP (Windows Satellite Programs) and developed and implemented the EasyComm protocol for those who wish to design their own rotor and radio controllers. Given Chris' experience supporting a variety of rotor interfaces in WiSP, the EasyComm protocol should contain the basic commands necessary to control rotors and radios. As a bonus, EasyComm is already implemented in several popular tracking programs such as WiSP, Nova, MacDoppler, and most recently, Mac and WinAPRS.

During the development of EasyTrak, it became evident that the EasyComm protocol was a good choice. It shielded the software author from knowledge of the rotor or radio. EasyTrak handled the translation from EasyComm command to rotor movement or radio control. For example, to move the azimuth rotor to 270 degrees, the following command is issued:

```
AZ270.0↵
```

Where "↵" can be a space, carriage return, or linefeed (the space allows multiple commands to be placed on one line). It is EasyTrak's responsibility to safely and accurately position the azimuth rotors to the commanded position. An example radio frequency command is:

```
UP145900000↵
```

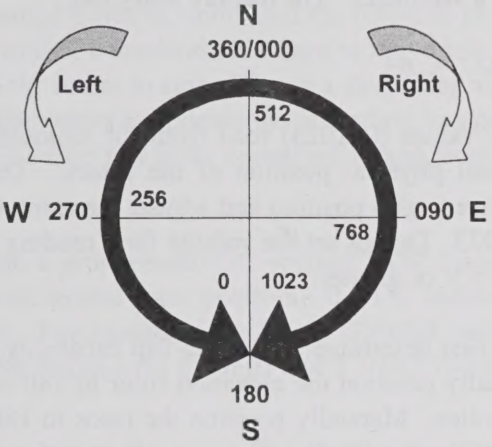
This will command the uplink frequency to 145,900,000 Hz (145.900 MHz). The software author does not need to know what radio model is commanded. EasyTrak handles all of the details. However, it became apparent that configuration commands would be necessary to tell EasyTrak about the rotors and radios interfaced to it. Thus the decision was made to extend the basic EasyComm protocol with configuration commands.

The goal of the extending EasyComm protocol for configuring EasyTrak required that the same basic command structure be followed (two letter command followed by a command value) and that it should not require the software author to implement the extended commands in order to configure EasyTrak. EasyTrak can be configured using a simple terminal program, and once configured, can be interfaced to any tracking program that supports the EasyComm protocol. The goal was to use the fewest commands necessary to configuration EasyTrak. Attachment A summarizes the EasyComm 2 and extended commands supported by EasyTrak.

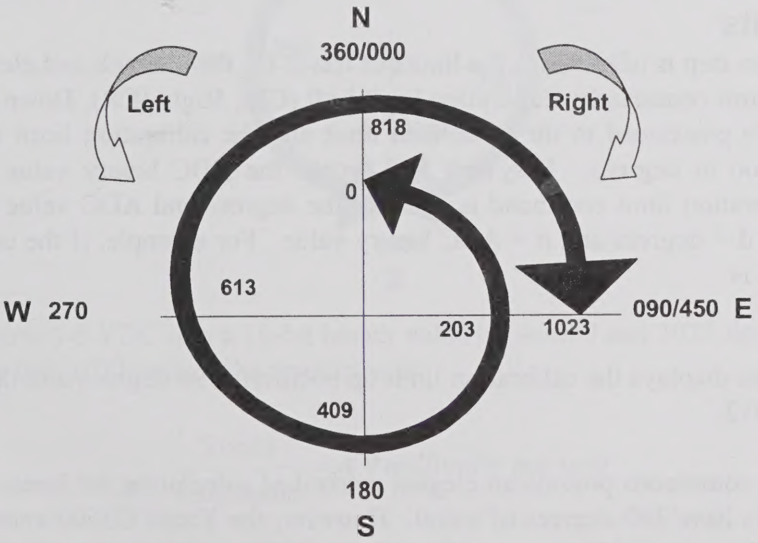
Rotor Configuration

The first rotor configuration command, RO, specifies the type of rotor combination. There are four choices: A = Azimuth only, E = Elevation only, C = AZ/EL Combination, or N = None (no rotor) (default setting). This command allows EasyTrak not to be tied to azimuth/elevation rotors only.

The second rotor configuration command, RS, specifies the physical stop of the rotor. There are two choices: S = South physical stop (such as the Yaesu G5400B/G5600B) or N = North physical stop (such as the Yaesu G5500). This command is necessary for EasyTrak to compute the physical position to the compass direction. The following diagrams explain why this is necessary:



The above diagram illustrates a south physical stop. The inner values represent the 10-bit ADC value that is read from the proportional 0-5 V analog signal supplied by the rotor controller. The outer values represent the compass directions.



In contrast, the above diagram illustrates a north physical stop (with 90 degree overlap) of the Yaesu G5500 rotor. Again the inner value represents the approximate 10-bit ADC value read from the proportional 0-5 V analog position signal supplied by the rotor controller. The outer values are the compass directions. These two types of physical rotor configurations require a different mathematical conversion. Once EasyTrak has been configured with the rotor type and physical stop, the rotors must be calibrated.

Rotor Calibration

First and foremost, the rotors must be assembled and calibrated according to the manufactures instructions. Once this has been done, the rotors can be calibrated to EasyTrak. The first step is to set the maximum 0-5 V analog position signal voltages (azimuth and/or elevation) coming from the rotor controller.

The CA command continuously reads and displays the ADC value for azimuth and elevation. This command precludes the need for a voltmeter. The display looks like:

```
AZ ADC = 127  EL ADC = 64
```

The numbers are the raw binary values (0..1023) read from the azimuth and elevation ADC channels. This number represents the actual physical position of the rotors. The user manually positions the azimuth rotor in the fully clockwise (right) position and adjusts the azimuth output voltage until the ADC value reads between 1020 and 1022. Do not set the voltage for a reading of 1023. That is the maximum value of the ADC and represents 5 V or greater.

To calibrate the elevation rotor, first determine if antenna-flip capability (0 to 180 degrees) or not (0 to 90 degrees) will be used. Manually position the elevation rotor to 180 or 90 degrees. For 180 degrees the above azimuth procedure applies. Manually position the rotor to 180 degrees and set the elevation ADC value between 1020 and 1022. For 90 degrees, manually position the rotor to 90 degrees and set the ADC value to 512 (half the maximum ADC value).

Calibration Limits

The second calibration step is to establish the limits of travel for the azimuth and elevation rotors. There are four calibration limit commands: **C**alibration limit **L**eft (CL), **R**ight (CR), **D**own (CD), and **U**p (CU). The rotor is manually positioned to the associated limit and the calibration limit command is entered with the rotor position in degrees. EasyTrak will record the ADC binary value associated with the position. If the calibration limit command is queried, the degrees and ADC value are displayed in the format CUd,n where d = degrees and n = ADC binary value. For example, if the calibration limit up is queried, the response is:

```
CU90,512
```

This example response displays the calibration limit up position is 90 degrees and the ADC binary value associated with it is 512.

The calibration limit commands provide an elegant method of calculating the linear travel of the rotors. The majority of rotors have 360 degrees of travel. However, the Yaesu G5500 azimuth/elevation rotors present a unique challenge with 450 degrees of travel (an additional 90 degrees of overlap between 0 and 90 degrees). The azimuth rotor is manually positioned to 450 degrees and the command CR450 is entered.

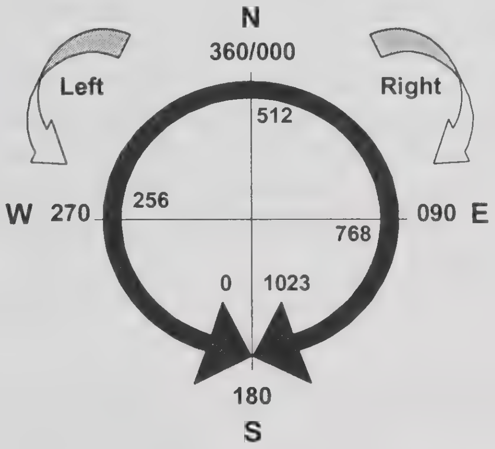
The calibration limit method also provides an elegant method of recording the minimum and maximum travel limits of the rotor. For example, if the rotor model is a Yaesu G5400B (which as a south physical stop) and the rotor installation has a physical obstruction between 180-270 degrees, a left most calibration limit can be entered. First, manually position the azimuth rotor to the desired left most limit, in this example, 270 degrees. Enter the command CL270. EasyTrak will not position the rotor any further left than 270 degrees to the physical stop at 180 degrees (to the left).

Design Implementation

The discussion up to now has been on the operation of EasyTrak. This section explains some of the technical details and design implementation of EasyTrak.

To position the rotors the commanded position is compared to the actual rotor position. If the rotor position is less than the commanded position, command the rotors to move right. If the rotor position is greater than the commanded position, command the rotors to move left. When rotor position is equal to commanded position command the rotors to stop. This is a simplified algorithm that in practice is a little more involved to implement. The rotors are positioned according to a proportional analog voltage that is digitized by the PIC ADC. This value has to be converted to and from degrees to be useful to tracking program.

The rotor indicates position with a proportional DC voltage. The Yaesu azimuth/elevation rotors use a linear 0-5 VDC signal to indicate actual rotor position. 0 VDC indicates the left most position of the rotor and 5 VDC the rightmost. For example, the Yaesu G5400B rotors (which have a south physical stop) use 0 VDC to indicate 180 degrees (to the left) and 5 VDC 180 degrees (to the right). Recall the diagram:



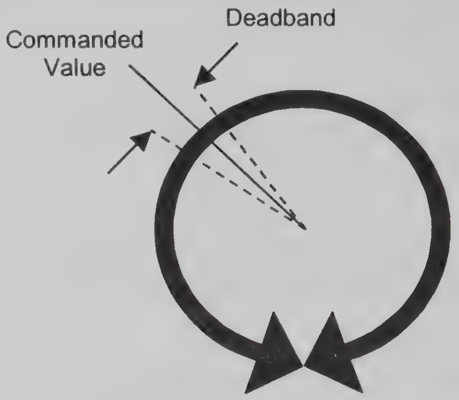
The PIC ADC converts 0-5 VDC into a 10-bit binary value between 0 and 1023 decimal. Effectively the ADC divides 5 volts into 1023 units. The resolution is:

$$\frac{5\,volts}{1023\,units} = 4.9\,millivolts\;per\,unit$$

The range of 0-5 VDC represents 360 degrees of rotation. The calculated volts per degree is:

$$\frac{5\,volts}{360\deg} = 13.8\,millivolts\ per \deg$$

The 13.8 millivolts per deg of rotation is a relatively small value. The rotor position voltage can be upset by noise (EMI on the sensing line and in the ADC). Plus there are mechanical limitations (gear slop) of the rotor and when the rotors stop moving there is a coast down period. Therefore, a boundary value must be specified to avoid the controller from continuously commanding the rotors left and right (chatter). This boundary value is called the deadband and it is a value (in degrees) specified by the user when configuring EasyTrak. The following diagram illustrates:



For example, if the commanded rotor position is 315 degrees and the deadband value is 5 degrees, the actual rotor position can be in the range of 310 to 320 degrees. This range appears to be a large value, but in practice, the rotors fall close to the commanded value. The deadband value should be set to a value that reduces the amount of rotor movement. This is an important property that deadband provides.

In satellite tracking applications, the tracking software is continuously sending azimuth and elevation position commands. The deadband specifies how often the rotor position will change. If the value is too small, the rotors will move often causing increased wear and tear on the rotor motors. If the value is too large, antenna pointing will not be optimal. The largest factor affecting the choice of deadband value is the antenna beamwidth. The positioning error budget is a combination of factors that can add or cancel each other out. These factors include:

Antenna Beamwidth
Rotor Gearlash
Nonlinearities in the position potentiometer
ADC Error
Electrical Noise

PIC Programming

Now that the operational and technical aspects have been determined, the next step is to write the firmware routines for the PIC. The C programming language was chosen for the size and complexity of the EasyTrak project. The basic program structure uses an interrupt service routine to buffer the serial data from the host computer and an infinite main loop comparing the commanded to actual rotor position and issuing the appropriate move and stop commands.

The interrupt service routine (ISR) buffers the serial data from the host computer. The ISR allows the main loop to continuous check command to actual several times per second. When a complete command has been received, it is placed in a buffer and a received command flag is set.

The infinite main loop checks if the received command flag is set for a new antenna position. If one has been received, the ASCII command value is converted to binary and compared to see if it falls between the calibration limits. Then the command value (in degrees) is converted to the corresponding ADC binary. All comparisons are done in ADC binary values.

The antenna position routine first determines the low and highband ADC values from the command ADC and deadband values. The following code segment illustrates:

```
lowband = cmdAZadc - AzDeadbandAdc;

if (bit_test(lowband, 15)         // lowband less than zero?
   lowband = 0;                   // correct

highband = cmdAZadc + AzDeadbandAdc;

if (highband > 1023)              // greater than max?
   highband = 1023                // correct
```

Once the low and highband values are determined, the rotor position is read:

```
antAZadc = readADC(AzChan);       // read antenna AZ ADC value
```

It is compared to the low and highband values to determine if the rotors should be commanded to move: Notice that the actual rotor position is compared to the deadband limits. If the actual rotor position falls inside the deadband limits, the rotors do not move. It is only when the actual rotor value fall outside the deadband limits that the rotors are commanded to move:

```
if (antAZadc < lowBand) {         // if antenna < lowband
   output_high(cmdRight);         // command right

} else if (antAZadc > highBand) { // else if antenna > highband
   output_high(cmdLeft);          // command left
}
```

Once the rotor is moving, it is desired that they stop in the center of the deadband. Therefore, the actual rotor position is compared to the command value:

```
if (antAZadc >= cmdAZadc) {       // if antenna >= cmd
   output_low(cmdRight);          // command stop
}

if (antAZadc <= cmdAZadc) {       // if antenna <= cmd
   output_low(cmdLeft);           // command stop
}
```

The above routine works for small antenna arrays and rotors that stop in a short amount of time. In practice, the Yaesu azimuth/elevation rotors stop very quickly. The above routine would have to be modified for larger antenna arrays and rotors that take a finite amount of time to coast to a stop.

The only difference between the azimuth and elevation routines is that azimuth needs to be corrected for north or south physical stop.

Radio Control

Radio control is not as complicated as rotor control. A frequency or mode command is received and sent according to the computer interface protocol of the radio. EasyTrak cannot change frequency bands on dual band radios. Very few (if any) have change band commands. Therefore, it is up to the operator to manually configure the radio prior to operation. Otherwise the radio will ignore the frequency command.

EasyTrak is configured by telling it the model radio used for the up and downlink. EasyTrak has two radio ports. This allows radio separates to be controlled for the up and downlink and it allows the control of dissimilar radio models. For example, a Kenwood TS711 can be configured to port 0 as the Mode J uplink radio (URTS711,0) and an ICOM IC475 can be configured to port 1 as the downlink radio (DRIC475,1). If a port is not specified in the radio configuration commands, it defaults to 0.

If the radio is a dual band model, the port value is the same for the Up and Downlink. For example, to configure an ICOM IC-821 dual band radio to port 0, the following commands would be entered:

```
URIC821,0
DRIC821,0
```

Conclusion

Design and development continues on the EasyTrak rotor/radio controller. Presently there are 8 beta units in test. Once testing is completed and finally design decisions are made, it is hoped to provide EasyTrak as a TAPR kit. Be cautioned, the information presented in this paper is subject to change as the design continues to develop. To follow EasyTrak developments see http://www.tapr.org/~n7hpr/easytrak/.

Acknowledgements

Many thanks to the EasyTrak development team who have contributed immensely to the EasyTrak design (in alphabetical order):

Don Agro, VE3VRW
Steve Dimse, K4HG
Dan Huton, VA3DH
John Koster, W9DDD
Lou McFadin, W5DID
Doug McKinney, KC3RL
Stacey Mill, W4SM
Keith Sproul, WU2Z
Paul Williamson, KB5MU

References

[1] Microchip Technology, Inc. http://www.microchip.com

[2] Keith Sproul, WU2Z, and Mark Sproul, KB2ICI, *WinAPRS/MacAPRS and Automatic Rotor Tracking of Moving Objects*, in this proceedings.

[3] The EasyComm 1 and 2 standard is documented in a text file "easycomm.txt" accompanying the WiSP satellite program available from http://www.amsat.org/amsat/ftpsoft.html#win-wisp

Trademarks and Copyrights

Easy Comm 2 and Extended Commands Supported by EasyTrak

The host computer issues a 2-character command identifier followed by a command value (if required). Commands are separated (terminated) by a space, carriage return, or linefeed. All commands are echoed back to the host computer.

Rotor Commands

Position Commands

Command	Meaning	Command Value	Example
AZn	**AZ**imuth	0.0..450.0	AZ10.2
ELn	**EL**evation	0.0..180.0	EL45.1
SA	**S**top **A**zimuth	none	SA
SE	**S**top **E**levation	none	SE
ML	**M**ove **L**eft	none – move rotor left 1-sec	ML
MR	**M**ove **R**ight	none – move rotor right 1-sec	MR
MD	**M**ove **D**own	none – move rotor down 1-sec	MD
MU	**M**ove **U**p	none – move rotor up 1-sec	MU

The AZ or EL command can be queried (AZ or EL command issued with no command value). The query will respond with the actual position of the rotors. For example, the query

AZ↵ (where ↵ is a space, carriage return, or linefeed)

will return

AZ270.0↵ (where ↵ is a carriage return/linefeed)

Deadband Value

Command	Meaning	Command Value	Example
DAn	**D**eadband **A**zimuth	0..9 (default = 5)	DA5
DEn	**D**eadband **E**levation	0..9 (default = 3)	DE3

Configure Rotor

Command	Meaning	Command Value	Example

ROa	**RO**tor selection	A = **A**zimuth only
		E = **E**levation only
		C = AZ/EL **C**ombination
		N = **N**one (no rotor) (default)
			ROC

RSa	**R**otor physical **S**top	N = **N**orth physical stop
		S = **S**outh physical stop (default)
			RSS

Calibration

Command	Meaning	Command Value	Example
CAn	**CA**libration	None	CA

The CA command is used to set the maximum output voltage from the rotor controller to an ADC binary value of 1020 to 1022. Entering the CA command returns a continuous reading and display of the ADC value for azimuth and elevation. Example display:

```
AZ ADC = 127  EL ADC = 64
```

Command	Meaning	Command Value	Example
CLn	**C**alibration limit **L**eft	0..450 (default = 180)	CL0
CRn	**C**alibration limit **R**ight	0..450 (default = 180)	CR360
CDn	**C**alibration limit **D**own	0..180 (default = 0)	CD0
CUn	**C**alibration limit **U**p	0..180 (default = 90)	CU90

The calibration limit commands are entered when the rotor is manually positioned to full left (CLn), full right (CRn), full down (CDn) or full up (CUn) positions. When the limit commands are entered, EasyTrak records the associated ADC value with the degrees n entered.

When the calibration limit command is queried, the degrees and ADC binary value are display in the format CUd,n where d = degrees and n = ADC binary value. For example:

CU90,512

displays the calibration limit up position is 90 degrees and the ADC binary value associated with it is 512.

The limit commands also record the minimum and maximum limits of the rotor. For example, if the rotor model is a Yaesu G5400B and the rotor installation has a physical obstruction between 180-270 degrees, the calibration command CL270 can be entered while the rotors are manually positioned to 270 degrees. EasyTrak will not position the rotor further left than 270 degrees to the physical stop at 180 degrees (to the left).

If the rotor configuration command RS (rotor physical stop selection) is changed, the CL and CR default values change as follows:

RSS – North to South physical stop:
CL180,0
CR180,1022

RSN – South to North physical stop:
CL0,0
CR360,1022

Radio Commands

Radio Frequency and Mode Commands

Command	Meaning	Command Value	Example
UPn	**UP**link Freq	in Hertz [Note 1]	UP145000000
DNn	**D**ow**N**link Freq	in Hertz [Note 1]	DN435000000
UMa	**U**plink **M**ode	CW, USB, LSB, FM-N, FM-W, NONE (default)	UMFM-W
DMa	**D**ownlink **M**ode	CW, USB, LSB, FM-N, FM-W, NONE (default)	DMFM-W

Note 1: Frequency band must be manually selected and the frequency value within the range of the band selected or the frequency command will be ignored.

Configure Radio Make and Model Selection

Command	Meaning	Command Value	Example

URr,p	**U**plink **R**adio	r = radio model (see below)
		p = port (0, 1)
			URTS711,0

DRr,p	**D**ownlink **R**adio	r = radio model (see below)
		p = port (0, 1)
			DRTS811,1

Current radio makes and models supported by EasyTrak are:

Yaesu: FT736, FT847
Kenwood: TS711, TS811, TS790
Icom: IC275, IC475, IC820, IC821, IC970
NONE (default)

If a port is not specified in the UR or DR command, it defaults to 0.

If the radio is a dual band model, the port value is the same for the Up and Downlink. For example, to configure an ICOM IC-821 dual band radio to port 0, the following commands would be entered:

```
URIC821,0
DRIC821,0
```

EasyTrak cannot change frequency bands. Very few (if any) computer controlled radios have change frequency band commands. Therefore, it is up to the operator to manually configure the radio prior to operation. Otherwise the radio will ignore the frequency command.

The benefit of having two radio ports is that EasyTrak can control dissimilar model radios. For example, a Kenwood TS711 can be configured to port 0 as the Mode J uplink radio (URTS711,0) and an ICOM IC475 can be configured to port 1 as the downlink radio (DRIC475,1).

Miscellaneous

EasyTrak Version Information

Command	Meaning	Command Value	Example
VE	**VE**rsion Request	none – returns version info	VE

Alarms

EasyTrak will alarm if the checksum of the configuration values held in EEPROM is invalid.

```
ALInvalid EEPROM Checksum
```

APRS Tiny Web Pages

Bob Bruninga, WB4APR
115 Old Farm Ct
Glen Burnie, MD 21060

Although APRS has revolutionized packet radio, it is also one step ahead of the next great leap in wireless technology which is handheld wireless access to Web pages. Although cell phones and pagers are just beginning to capitalize on this next wave of technology, *APRS already has it*! Take a look at the screen of your Kenwood TH-D7 or TM-D700... What you see are hundreds of Tiny Web Pages of information!

In this context, these radios already display over 200 pages of station information, 32 pages of Messages and 20 pages on the DX list on your dashboard or in the palm of your hand. The purpose of this paper is to show how we can exploit this display capability to give mobile and handheld APRS users wireless access to a wealth of on-line information. This paper will cover the following topics:

- The existing 252 Tiny Web Pages that can be displayed
- How we added Satellite tracking and schedule data using "push" techniques
- How we expanded to dozens of other categories based on "query" techniques
- The formats for using the DX list to display Tiny Web Pages
-

Photo 1. This photo shows W4HFZ's mobile APRS installation. Using the Kenwood APRS mobile data radio, even without a laptop, he has access to hundreds of pages of display information. Now we are adding Tiny Web Pages and whole new databases to these displays.

EXISTING STATION PAGES (using "push" technology)

First I will review the appearance of the conventional usage of these Tiny Web Pages. These existing pages can be considered to use "push" technology, in the sense that you do not have to request them. Just turning on your radio and monitoring for a few minutes will update these 252 Tiny Web Pages automatically in your radio. In the following examples, I will use the screen of the TH-D7 HT to illustrate the Tiny Web Page. The TM-D700 is similar but the pages are larger and contain more text, so by using the HT in these examples you will see the worst case display scenario. The STATION pages follow 5 basic formats. The first page is just the list of stations:

STATION LIST

```
+--------------+
|  1:WB4APR    |
|  2:WU2Z      |
|  3:KB2ICI    |
+--------------+
```

The Station list shows three calls at a time and you can scroll up or down the list through the entire 40

On selecting any station, you can cycle through 4 additional pages:

```
STATUS PAGE       ICON DISTANCE     LAT/LONG          CSE/SPD/COMMENT
+--------------+  +--------------+  +--------------+  +--------------+
| 1:WB4APR     |  |  1:WB4APR    |  | 1:WB4APR     |  | 1:WB4APR     |
|   To Alabama |  |    FM19dx    |  | N 38 59.11   |  | Enroute      |
|   for XMAS   |  |    16.4mi -> |  | W 76 29.11   |  | cse284°s055m |
+--------------+  +--------------+  +--------------+  +--------------+
```

Yet, these 4 formats are not limited to only Mobiles, but ANY station or object. Even in conventional APRS we use these pages every day to look at these kinds of information:

- CALLSIGN and SSID
- Station Type (Fixed, Digi, Moving, THD7, TMD700, Mic-E, WX etc)
- Station ICON (Over 300 but only the most common 16 are graphical)
- Grid Square
- Distance and Bearing from your present location
- Location to nearest 60 feet (Latitude/Longitude)

THE 5TH CUSTOM PAGE

In addition, each of these stations has additional information on its 5th page depending on the type of station as shown below:

STATION TYPE	DISPLAYS
HOME STATION	XMTR Power, Antenna Height, Antenna gain, Directivity
WX STATION	Wind Direction and speed, Temp and Rain in last hour
MOBILES	Station type, Course and Speed, and comment
OBJECTS	Who posted it, course and speed, comments, etc
SATELLITES	Uplink and downlink frequencies, and present Doppler
HURRICANES	Who posted it, Wind Speeds, pressure, Course and Speed

PAGE SELECTION AND MESSAGE PAGES

We can SELECT these pages by moving forward or backward in the list or by setting the POSITION LIMIT so that we capture only stations within X miles of our current location. Thus we have complete control over our initial selection of these STATION Tiny Web Pages. Further, we can use standard APRS queries to query ANY station to report its pages or we can directly query any of these stations for additional information about their station or operation. These responding Tiny Web Pages are in the form of two page messages. Typical Query Responses from APRSdos stations look like this on the D7's Tiny Web Pages:

1st MESSAGE PAGE   2nd MESSAGE PAGE:

```
+--------------+  +--------------+
|  >WB4APR     |  |  >WB4APR     |  Response to a QUERY DIRECTS lists all
|   Dir= WU2Z  |  |   K3FOR N8PK |  stations heard direct by WB4APR
|   K4HG KH2Z  |  |    KB2ICI W4 |
+--------------+  +--------------+
+--------------+  +--------------+
|  >WB4APR     |  |  >WB4APR     |  Response to a QUERY HEARD shows number
|   Hrd KB2ICI |  |   2 17 16  8 |  of packets per hour for the last 12 hours
|   15 23  9   |  |    14 23  4  |
+--------------+  +--------------+
+--------------+  +--------------+
|  >WB4APR     |  |  >WB4APR     |  Response to a QUERY TRACE shows the path
|   Path= WB4A |  |   >K3ATI-12> |  from WB4APR to requesting station
|   PR-11>N8PK |  |   K3XY-3>KB2 |
+--------------+  +--------------+
```

DX LIST

In addition to the Station list and the Message list, the radios also have the DX LIST which is used to capture DX spots as monitored from any DX cluster Frequency. These pages are 10 deep by two wide and display all the useful information provided in these DX spots. Unlike the other pages, however, these pages are volatile and are lost on each power down. This is usually not a problem due to the real-time value of these posts. It is the volatility of these DX LIST pages that make them attractive for some Tiny Web Page applications.

SATELLITE TRACKING PAGES and APRSdata.EXE

One of the first extensions of this Tiny Web Page concept (still using "push" technology) was to use the APRS network to post satellite tracking data to all mobiles and handhelds whenever one of the FM satellites comes into view. This is done by regional copies of my APRSdata.exe program that can transmit this information as a moving object to all stations within range. These Objects then provide the users with everything they need to know about the satellite right on their HT's Tiny Web pages as shown below:

```
+--------------+  +--------------+  Satellite-in-view PAGE
| 1:AO27       |  |  1:AO27      |  Showing Up and Down freqs plus doppler
|   145.85  -2 |  |    FM18RX    |  Showing Range and direction
|   435.795 +6 |  | *  1028mi -> |
+--------------+  +--------------+
+--------------+  +--------------+
| 1:AO27       |  | 1:AO27       |  Shows current position and its
| N 38 59.11   |  |   OBJECT     |  course and speed so you can tell
| W 76 29.11   |  |   c247 s999  |  if it is approaching or departing
+--------------+  +--------------+
```

By watching the Range page, or the amount of Doppler, the handheld user can tell when he can work the satellite. When the satellite gets to within about 1000 miles, you can usually work it from the mobile or the HT with your whip antenna.

SATELLITE SCHEDULES

But since these satellite objects are only usable if the user happens to be on the air at the time, the APRSdata program also provides a single Satellite Schedule every 10 minutes indicating all satellites that will be in view in the next 80 minutes. Since these schedules are repeated, it was decided not to displace good data in the POSITION or MESSAGE lists, but to write these schedules to the volatile DX LIST. These pages, although somewhat formatted to match the format of a typical DX spot can be used for displaying almost any kind of information. They are ideal for this scheduling application. Here is how these satellite schedules appear on the DX LIST:

```
+------------+   Future PASS predictions updated every 10 minutes
| UO22@1435  |   as listed in the D7's DX list showing all FM satellites
| SO35@1453  |   expected in the next 80 minutes
| UO14@1532  |
+------------+
+---------------------------+
| 1:UO22@1435          SATS | Here is the same info displayed on the D700
|            UO14@1443      | which can display four satellites worth of
|   KO25@1455 SO35@         | schedule data on one screen.
|   1510                    |
+---------------------------+
```

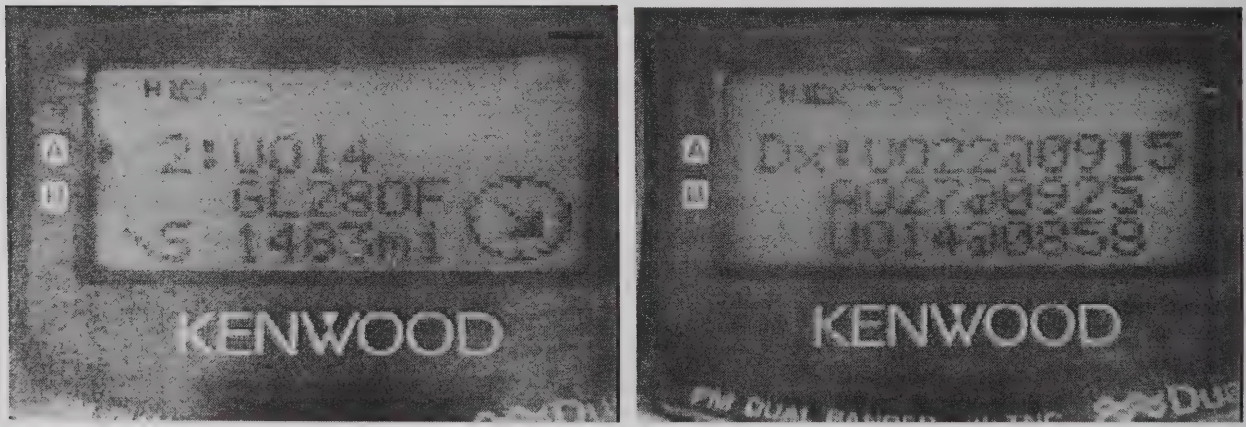


Photo 2. The left photo is just one of the 4 Tiny Web Pages that alerts the user that UO-14 is in view and is 1483 miles to the Southeast. Others show the Up and Down frequencies, position and direction it is headed. The right photo shows the Satellite Schedule for the next 3 satellites as it appears on the DX list.

GENERAL PURPOSE TINY WEB PAGES

The satellite Tracking Objects and 80 minute Schedules were just one of the first applications for the APRS Tiny Web Pages. As soon as we saw the power of this technique to *PUSH* information to the users, we realized that there could be an infinite depth of information that could be made available to *users ON DEMAND*. Just like the real WEB, users transmit QUERIES to the APRS system and the data is returned in any of the above variety of formats. All it takes to provide this service is someone in each major APRS area to run a copy of APRSdata.exe on an old PC connected to a TNC. Users in the area need no a-priori information to use the system. It will respond to Queries sent to the generic callsign of QDOS. The following two lists, then, summarize the information available to local users via their Tiny Web Pages. First there is the information already available on every APRS network:

- The location, range and bearing of the last 40 APRS stations around you
- The location, range and bearing of all DIGIpeaters in range
- The local WEATHER including wind speed, direction, rain and temp
- ANY APRS bulletins
- Any APRS mail (think of this as Email)
- REAL one-line EMAIL from the radio TO anywhere

Next is the new data added to the network by local APRSdata servers.

- A Satellite Schedule every 10 minutes covering the next 80 minutes
- Satellite positions every 1 minute when in view
- Direction and range to the satellite from YOUR location
- Satellite Uplink and Downlink Frequencies
- Approximate Satellite Doppler for tuning

Finally are all the new data added to APRSdata and now available on demand by users from any local station running APRSdata information server. Just send a Message to QDOS with these Keyword Queries:

• HOSP	Nearest Hospital
• RS	Nearest Ham Radio Stores
• NWR	Nearest NOAA WX Radio xmtrs and freqs
• NWS	Nearest Nation WX service sites
• VOICE	Nearest wide area Travelers VOICE repeaters
• ATV	Nearest ATV repeater or Shuttle retransmission repeater
• CRASH	Nearest old Aircraft CRASH sites (to avoid errors during SAR)
• CAMP	Nearest Campground
• FD	Nearest Field Day Site and talk in freq
• CLUB	Nearest Ham Radio Club and meeting schedule
• NET	Nearest Net and frequency

Other data files that have been suggested:

- PD = Police Station (Local/Municipal) location and telephone number
- SP = State Police Station/Barracks locations and a telephone number
- EOC = EOC Locations and local government OEM freq.
- BBS = List of local 2M packet BBS, location, freq., etc.

The following examples show the how some of this information is delivered to the TH-D7 handheld display in response to a Query. The first screen is the 10 second "immediate" screen that summarizes the data the instant the packet comes in. After that, it is stored in the STATION LIST along with the last such 40 items. Each such ITEM has actually 5 "Tiny-Web-Pages" of detail. Only the first two are shown here. The others show exact LAT/LONG, and CSE/SPEED and other data.

```
IMMEDIATE PAGE    RECALL PAGE
+--------------+  +--------------+  In response to a CLUB query
| AARC         |  |  1:AARC      |  The   is an actual ICON graphic
| 3rd Thurs    |  |     FM18RX   |  and the -> is a compass rose
| 147.105      |  |     13.6mi -> |  showing the direction
+--------------+  +--------------+
+--------------+  +--------------+  Location of the nearest Radio Shack
| RS           |  |  3:RS        |  in response to a RS query.
| OBJECT       |  |     FM18CD   |
| fm:WB4APR    |  |     3.6mi  <- |
+--------------+  +--------------+
+--------------+  +--------------+  In response to a NWR Query
| NOAA         |  |  1:NOAA      |  shows nearest NOAA WX radio station
| 164.250      |  |     FM18RX  | |  and frequency
|              |  |   13.6mi  v |
+--------------+  +--------------+
+--------------+  +--------------+  Each home WX station transmits these
| N8ADN        |  | 5:N8ADN      |  every 10 minutes or so, or you can
| WEATHER      |  | dir015  s13m |  do a Query to get the closest one to
|              |  | t043F r0.01" |  you at any time...  It shows WIND speed
+--------------+  +--------------+  and direction, TEMP and RAINFALL
```

APRS TINY WEB PAGES FORMAT

The existing APRS formats can be used to write any POSITIONAL or TEXT type information to the STATIONS or MESSAGE lists. But when you write to either of these pages, you may be overwriting information that the user wants to retain. This is why we find the DX LIST pages to be the target for most of our text information. These DX LISTS are volatile and are not retained whenever the radio is turned off. Thus, they are perfect as a scratch pad display area for our Tiny Web Pages.

Back in 1998 I re-defined this DX list as an "APRS RESOURCE" (See README\RESOURCE.TXT in all APRS docs) and defined how we could use this page to display "resource" information of use to mobile users. The following examples show how the Kenwood Radios parse and display the DX SPOT format. First is the on-air packet format:

```
URCALL>RESORC:DX de YOURCALL.>FIELD-ONE.ITEMNAME***FIELD-TWOFIELD-3333
              ***********FORTH
```

Notice that due to a program quirk in the original TH-D7, that the AX.25 TOCALL must NOT be any of the normally recognized APRS generic calls. Thus we chose RESORC as the target call for this kind of packet. Beyond this filter, the Parsing strings are the characters "DX de ". Once these are recognized, the remaining 59 characters are parsed into five separate fields and displayed as shown below, two pages for the D7 and one page for the D700. The 14 characters shown as asterisks are ignored since they do not fit on the screens.

```
+--------------+      +---------------+
| Dx:RESOURCE  |      | Dx:RESOUIRCE  |
| FIELD-ONE    |      |    FIELD-TWO  |
|       FORTH  |      |    FIELD-3333 |
+--------------+      +---------------+

And on the D700 as:   +----------------------------+
                      | 1:RESOURCE           FORTH |
                      |            FIELD-ONE       |
                      |   FIELD-TWO FIELD          |
                      |   -3333                    |
                      +----------------------------+
```

APRSdata.EXE TINY WEB PAGE FILE SERVER

The program APRSdata.EXE actually began as a satellite tracking program called APRStk.exe. This program was simply the addition of satellite prediction routines to normal APRSdos to predict the positions of all the moving satellites and to TUNE the new Kenwood radios automatically to any satellite that was in view. As soon as we put it on the air, we realized that we could use these same routines to serve out the Satellite objects and Schedules to local Mobiles and Handheld users as well. Then we quickly realized that ALL of the normal APRS Map Overlay Files built into every copy of APRSdos could also be made available on demand using the same techniques. This then evolved into the present APRSdata.EXE program. But it should be noted that ALL of the Tiny-Web Page positional information is maintained in the same original XXXX.POS format already used by APRSdos and other versions for map overlay display of local information. Thus, anyone can prepare ANY kind of Tiny Web Page Query/Response simply by editing all of the positional data into a .POS formatted file and giving it the NAME of the QUERY to be used on the air. For example, here are a few sample lines from the CLUB.POS file for all the clubs in our local area:

```
Mobileers!3905.46N/07637.33W/?nd Friday 146.805
AARC !3853.87N/07639.31W/3rd Thurs 147.105
MMARC!3911.60N/07640.87W/Unknown   145.13
USNA !3858.88N/07628.88W/Noon Tues 147.105
ARINC!3858.45N/07633.40W/Unknown   147.105
GARC !3859.92N/07650.79W/Noon Wed  146.835
SMARC!3844.  N/07659.  W/2Fri 1930 147.15
LARC !3906.  N/07651.  W/4th Wed   147.225
NCDXA!3900.  N/07700.  W/Sund 2000 147.000
AARC !3851.  N/07708.  W/Tues 1930 145.47
BARC !3924.  N/07640.  W/Tues 2030 146.67
CARA !3913.  N/07653.  W/4Tue 1930 147.135
GMRA !3900.  N/07656.  W/SAT  1130 146.88
```

The first variable length field is the NAME of the item. Next is the fixed format LAT/LONG and ICON. Then up to 20 characters of additional information for display. By aligning these characters in fixed columns, they appear nicely on the Tiny Web Pages on the TH-D7 HT.

NOTES

I chose QDOS as the TO-CALL key word that triggers the response from APRSdata since it is a DOS program. Unless or until other programs evolve with this capability, it will also respond to the key word "QUERY" as well.

Second, the initial versions of APRSdata respond only with the closest item of the given category. In many cases, you may want to have a second opinion or to see the second or nth closest similar item. In later versions I hope to add this ability by simply following the Query KEYWORD with a number. Thus CLUB gets you the closest club. But CLUB 2 gets you the SECOND closest CLUB and so forth.

Third, APRSdata sites should be reasonably spaced. It is OK to have minor overlaps, because a user half way between two APRSdata servers would then get TWO responses, one that is closest to him known by APRSdata server A and another response that is closest to him in the direction of APRSdata server B.

SUMMARY

Thus, using the existing APRS formats and the DX Spot format, you can write almost an infinite variety of Tiny Web Page information to mobile and handheld users. Be thinking how you can apply these techniques to your local area. Remember, none of this has any value unless someone in YOUR area prepares the various data base entries in the first place. So get out those databases and start building your .POS files for anything you can think of that a mobile or handheld user might want.

Will these Queries overload the APRS frequency? No. Each such query response is only a SINGLE 1 second packet. (Locally generated and managed.)

Also, Home stations don't need any of this. They already have access to all of it either in their respective APRS programs or by access to the WEB. These TWP's are only for the mobile who does NOT have a running APRS laptop. THus it is a small number of users that will be requesting data at any time. Even during COMMUTER hours, less than 10% of all stations on the air are mobiles...

Oh the fun we will have building these TWP databases of AMATEUR RADIO RELATED or HAM TRAVELER data for users...!

For more information see http://web.usna.navy.mil/~bruninga/satinfo.html

Intelligent Digipeating using DIGI_NED on Obsolete PCs

Henk de Groot, PE1DNN
Schopenhauerstraat 10,
7323 MC Apeldoorn, The Netherlands
pe1dnn@amsat.org

Abstract

In Europe the roll-out of APRS is well under way right now. However, we do not have a stable APRS infrastructure. This paper describes how we use old and obsolete PCs, together with packet hardware commonly used in western Europe, to augment the APRS infrastructure.

Key words

DIGI_NED, digipeater, intelligent, PC, DOS, Linux.

Introduction

At the end of 1999 we experienced the re-birth of APRS in Europe. Introduction of APRS had been tried before, but it never really took off. Now it is different. All packet and RTTY bulletins published by different clubs have some feature articles on APRS. Also PWGN, the Dutch Packet Workgroup, which previously supported the "traditional" packet network, now runs articles about APRS in their magazine. At every hamfest in the last few months there has been a demonstration of APRS.

The situation concerning packet in Europe is quite different from the US. Here we have a reasonably efficient packet network, all connected by radios I must add. Bulletins and private messages are almost guaranteed to arrive. Most of the nodes, BBSs and DX clusters do not use TNCs. The TNCs that are used are almost always fitted with NORD><LINK's "TheFirmware" and operate in KISS mode. FlexNet TNCs use a special 6Pack EPROM to give better control over the radio.

The stations that do not use TNCs (and that is the majority) use special SCC cards fitted with Zilog's Z85x30 chips. Connected to these cards are a variety of modems, from TMS3105-based 1k2 modems, G3RUH 9k6, Cadams and DF9IC (both German modems) to real high-speed modems (from 19k2 through 2 Mbps) for interlinks. All nodes in this area support at least 9600 bps FSK. Besides that, German stations use RMNCs which are FlexNet nodes using dedicated hardware.

The end users – the individual stations – do not use TNCs either (or if they do, they only use KISS mode). Most of them use cheap modems like BayCom's 1k2 modem (and clones), Par96, PicPar and IV3NWV's YAM modem (this one I use myself). Power users also use SCC cards for their private station.

Finally we in Europe, at least here in the Netherlands, do not have the same emergency tasks as in the US. Western Europe is densely populated, with an excellent communications infrastructure almost everywhere. Furthermore, we are not bothered by earthquakes or tornadoes like in the US. This means that APRS is really a "fun" project, and the "emergency" part of it is much less important.

This is the background against which our involvement in APRS started.

How we started with APRS

About one year ago Remko (PE1MEW, a nearby ham) and I had some thoughts about having weather measurement stations all over the country, collecting and exchanging WX telemetry. At that time it was only an outline plan, and life went on. Later, when we started to look at APRS, we realized that this was a suitable exchange mechanism for just this kind of data. If APRS were used we would also have the clients to display the weather data, which is a huge advantage.

For this to work there have to be enough digipeaters, so when one disappears and another appears the network remains usable, as long as there are at least a critical number of digipeaters on the air.

Our first objective was to create a tight closed network. For this we needed very cheap digipeaters that run reliably. Given the history in Europe, the most logical way to do this was to use PCs, with cheap and obsolete 80286-class PCs being more than adequate. Since many people have migrated to 9k6 over recent years, there are now many BayCom 1k2 modems which are no longer used, and which could be applied again for APRS. Also for cross-band work SCC cards should be supported as well as the good old KISS modem.

We need the cross-band functionality. In Europe the major APRS frequency is 144.800 MHz. In the Netherlands novices are not allowed to use digital modes on 2m. They are allowed to use digital modes on 70cm however. So cross-band functionality between 2m and 70cm is needed to let novices join in the fun.

After evaluation of existing digipeaters we found none would do what we wanted; the digipeater should be "intelligent", conform to the APRS specification and be stable. Besides that, we also wanted source code so we can add the features we want, like support for home-brew weather measurement equipment.

The plan

This is how the plan was born to build a new digipeater. The major requirements for this new digipeater have already been mentioned. In addition, the digipeater should be completely configurable and should not have hard-coded APRS knowledge. All configuration parameters should be soft-coded in an initialization file. This is important because every station is different. There was one program, StealthDigi, that already worked that way. StealthDigi itself was however not reliable and could not do what we wanted our digipeater to do, but the basic idea was okay and was adopted for our project.

To keep the access to the hardware simple and not to reinvent the wheel, we decided to base the digipeater on TFPCX. TFPCX is a popular program in Europe and is used as software TNC for the cheap and simple modems, for SCC cards and for KISS TNCs. Except for sound-cards, the use of TFPCX would cover all popular packet hardware.

That is how I started to write the program. After some experiments, however, I found that TFPCX could not be used. An application on top of TFPCX is either the originator of a packet or the receiver, but not some station in between – there is no access to the digipeater list to allow application-controlled digipeating. So I took TFPCX apart and reused only the lowest layer. Advantage of this was that this part of the code was covered by the General Public License, whereas the rest of the code was copyrighted by ALAS – a more restricted public license.

The low-level TFPCX part became a simple resident program that sends and receives raw AX.25 frames. On transmission the driver adds the CRC and transmits it to whatever hardware is specified. On reception the driver verifies the

CRC and offers the frame to the application. The driver still supports all the hardware that TFPCX supported, and just like with TFPCX, 8 ports can be used simultaneously.

The application

After solving access to the hardware, the digipeater application was built. The name of the digipeater is "DIGI_NED": "DIGI" because it is a digipeater and "NED" for "Nederland" which means "The Netherlands" in Dutch – showing some pride in our country…

The application runs under DOS, because of the 80286 constraint. The system can easily be put onto a floppy, saving the need for a hard disk. Although all the digipeating rules are soft-coded in the initialization file, users subsequently came up with scenarios and problems that were not covered at first. So capabilities to support those new requirements have been added.

I chose to release the digipeater as Open Source. As I said in the introduction, APRS is a fun mode here in Europe, so as many people as possible should be able to enjoy it. But beyond that, we ran into the problem that no suitable digipeater code was available to realize our ultimate goal, to build weather stations. So if other groups have similar ideas they now have access to the DIGI_NED code to build whatever they want for their project. A nice spin-off would be that we also might enhance our own solution with third-party designs in the future.

The driver part of DIGI_NED, which was taken from TFPCX, is available as Open Source too. That means that when you want to build some DOS program that needs low-level access to AX.25 and should support a lot of hardware, you can take it. Since it is a separate TSR you can even bundle this GPL product with non-GPL stuff.

Linux

During development I had the need to test the digipeater with multiple ports. I had already played around with the Linux AX.25 tools and libraries, so I decided to try to run DIGI_NED under Linux. After some study of sockets programming and looking at the solutions used in other AX.25 utilities (especially net2kiss) it turned out to be rather simple to add Linux support. The rest of the code ran first time after I stubbed some DOS specific calls. In fact, from that moment on I developed the product under Linux. Before each new release I test DIGI_NED under DOS, and up to now it has always worked first time. I can now test the digipeater under Linux using internal loop-back links. This way I can control the runtime environment and do not transmit bogus packets on the air.

So what do we have now

What we have accomplished now (and I say "we" because although I wrote it, I have incorporated much user feedback) is a state-of-the-art digipeater. Because the PC has so much more power than a TNC it can do more than any TNC on the market. And remember that the development started only in February 2000, less than 6 months of spare time. By the time you read this I'm sure more features will have been added.

Capabilities of the current version include:

- Low hardware demand; for DOS a 80286 PC is more than enough. I'm almost sure that it will even run on a 8088 – it needs a recompile for this. For Linux, any Linux-capable PC will do. I have not had the opportunity to test this on other non-PC Linux machines (may work nevertheless).
- DOS & Linux capable:
    - AX25_MAC driver for DOS (included)
    - Uses Linux AX.25 kernel interfaces

- Rule driven:
  - The only APRS knowledge in the digipeater code is APRS message handling. Behavior for APRS digipeating is completely defined by the digipeating rules.
- Normal digipeating:
  - RELAY, TRACE, WIDE
  - <Fill in your own, it's all configurable!>
- Intelligent digipeating:
  - TRACEn_N, WIDEn_N
  - Digipeating on SSID
  - <Fill in your own, it's all configurable!>
- Call substitution:
  - <Fully configurable>
- Extremely configurable:
  - <Everything>
- Drives up to 8 ports with different types of hardware:
  - BayCom 1k2, PicPar, Par96
  - OptoPcSCC (PA0HZP), PE1PET SCC, BayCom USCC, DRSI, FSCC
  - YAM 9k6 modem
  - KISS
  - BPQ-Ethernet
  - All Linux devices that are supported by the kernel.
- Digipeats frames with AX.25, IP, ARP and NETROM PID.
- Duplicate checking:
  - Remembers source call and CRC of the payload
  - <History time configurable>
- Mheard list:
  - Keeps call, heard date and port
  - Mheard list size configurable
  - Query on port number
  - Query on call
- Preemptive digipeating:
  - Takes packets out-of-order
  - Keeps or removes intermediate calls
  - <Fully configurable>
- Support for "local" ports:
  - To fill in "black spots", ports can be defined as "local"
- Message capability:
  - Some predefined built-in messages
  - Response to standard ?ping?, ?aprs?, ?aprst, ?aprsd and ?aprsm queries
  - Fully configurable roll-your-own queries, make TinyWebPages!
- Blocking of unwanted calls:
  - NOCALL, N0CALL, MYCALL
  - <Define your own, fully configurable!>
- Remote exit for remote maintenance:
  - Exit the digipeater and start a program for uploading new versions; we use NetCHL for this.
  - Remote reboot of system
  - Ability to switch off these remote functions.
- Logging:
  - Log digipeater actions for debugging
- Availability:
  - DOS-executable package
  - Sources for DOS and Linux (same source)
  - General Public License

Conclusion

Undoubtedly I have forgotten to mention some of the features – there is so much that has gone into this program in such a short time. With DIGI_NED you will have the most sophisticated APRS digipeater available today. If you disagree let me know and I will add the missing feature. DIGI_NED was initially stable for 1½ months in a row, then it was interrupted by an upgrade. New versions with added functionlity have since been coming out about every two weeks or so.

DIGI_NED is delivered with documented sample files and manual.

Are there disadvantages? Yes. First of all you need a PC, and even if it is a small and cheap one, it will always be physically bigger than a TNC. Another disadvantage is the flexibility, which on one hand makes it possible to do what you want it to do but also makes understanding the complete product not so easy. The sample files will provide a good start. If you want to use the product's specific features you have to understand what the digipeater does and how it fits in your local situation.

References

- APRS is a registered trademark of Robert Bruninga, WB4APR, reference: http://web.usna.navy.mil/~bruninga/aprs.html
- DIGI_NED can be found on the Web on the VrzApDxw pages: http://www.homepages.hetnet.nl/~remko
- APRS Protocol Specification version 1.01: http://www.tapr.org/tapr/html/Faprswg.html

NEW MILLENIUM MORSE

by Roy Ekberg, WØLIQ & Martin Schroedel, K9LTL

ABSTRACT
This covers aftereffects of 5 WPM Morse on Computer Assisted Communication (CAC) system proposed in the 17th ARRL and TAPR Proceedings. A prototype in ARRL's Library is indexed C-135-1. R & D on this began in 1989.

KEY PHRASES
Encrypted message benefits, Digital Shorthand messages, improved Q-code system, international CAC system standards

CAC SYSTEM AND 5 WPM MORSE CAN OFFER HAMS BIG BENEFITS

Many hams wonder why they need Morse anymore, at what rate of sending that it should be exchanged, etc. Such talk preceded FCC's 5 WPM CW test ruling. Controversial arguments alleged Morse had become obsolete! We disagree. Modern definitions of word "code" remind about code's typical benefits. Web←ster's Encyclopedic Unabridged Dictionary (copr. 1996) offers this definition:

> 17. a. the system of rules shared by participants in an act of communication, making possible the transmission and interpretation of messages; b. (in socialistic theory) one of two distinct styles of language use that differ in explicitness and are sometimes thought to be correlated with differences in social class.

Hams do enjoy a "class distinction" bestowed by governments. To enjoy this, they must pass license tests and specify their station's locations precisely. Hams often send short Q-codes for "explicit communications." These codes are also very useful for foreign contacts. Actually, hams didn't invent Q-codes...they adopted many from those used by maritime services. We presumed this after reading old codes in ARRL's 1940 pre-WWII handbook. Three examples are as follows:

QTI	What is your true course?
QTJ	What is your speed?
QUJ	Will you indicate the true course for me to follow, with no wind, to make for you?

Code QUJ is a classic example of "encoded explicitness" which could also be sent instead as "123." in digital form (with far less dots & dashes)!

Modern technologies expand the text and graphics which can be stored and downloaded via remote-control type digital systems. Our question now is: Can digital systems be kept updated periodically despite rapidly advancing technologies? FCC forbids encryptions presently, but proposed encryptions would be in the public domain. Another important question we ask is: Can digital systems be made for aiding hams who have disabilities like hearing loss or blindness?

To keep future system's costs low, we presume publishers (especially ARRL) would be permitted to include advertizer type information in software, manuals, etc. for digital cross-referencing. That would benefit hams by allowing them to show pictures of their equipment or discussing various technical features, etc.

We are deeply indebted to the mariners who appreciated and invented the construct of encrypted messages. Those lent practical ways to communicate during normal times and times of emergencies. Rather than abandoning useful encoded systems, hams should seek to modernize them to enjoy the benefits they can offer us!

Channel Capacity Simulation of Peer-to-Peer Spread Spectrum Satellite Transponders.

Matthew Ettus, N2MJI
matt@ettus.com

August 7, 2000

Abstract

A spread spectrum transponder for the International Space Station has been proposed, the goal of which is to provide medium to high bit rate digital communications to radio amateurs. In order to help predict the capabilities of such a system, as well as aid in its design , a simulation of system bit error rates has been performed. The simulation is performed at the channel symbol ("chip") level, using a freely available communications system simulator package. The results of these simulations show that over a hundred users may participate in digital voice communications at a time using the transponder.

1 Introduction

The Spread Spectrum Wideband Transponder[1] (SSWBT) is designed to be a payload on the EXPRESS Pallet of the International Space Station (ISS). The SSWBT is currently in the proposal stage. Its goals are to simultaneously provide:

- Low cost digital voice communications
- Digital Videoconferencing
- High bitrate, low-latency data transfer
- Development of Spread Spectrum technology in the amateur community

One of the greatest features of the SSWBT concept is that while more complex and expensive systems with high power and gain will be necessary to transmit at the higher bitrates, nothing extra will be necessary to receive these transmissions. Thus, the low bandwidth systems, besides being useful for voice communications between comparable users, can be effectively used for such applications as web browsing, and file retrieval. Ten kilobits per second is plenty of bandwidth for requesting web pages, which would be served by the medium or high bandwidth systems. Highly asymmetric links are very useful for these applications

2 SSWBT Design

2.1 Features

- Direct Sequence Spread Spectrum (DSSS) Modulation
- 2.4 GHz Band Uplink

- 5.7 GHz Band Downlink
- 5-10 MHz wide signal bandwidth
- Automatic Power Control
- Scalable bitrate

2.2 Capabilities

This system will be able to accommodate over 100 simultaneous digital voice users, several high bitrate video conferencing sessions, and a high-bandwidth data link, all at once. Stations within 400 miles of the point directly below the ISS will be able to access these facilities, providing a coverage area of about half a million square miles. It can provide high data rate, asymmetric data links to small mobile users, with tiny patch antennas. User systems will have low power consumption.

2.3 General Architecture

In order to receive and demodulate SS signals from hundreds of users at one time, hundreds of demodulators would be necessary on the ISS. Instead, the SSWBT simply amplifies and retransmits the signals which it receives. This allows the ground stations to each pick out and demodulate their own signals. A key advantage of the SSWBT is its very simple space segment. The payload will consist of a linear transponder, and a simple "carrier" signal generator, which will actually generate a known SS signal. All of the complexity will be in the ground stations. This allows for easy changes to the modulation format, and avoids the need for complex and expensive radiation-hardened DSP components.

2.3.1 Modulation and Coding

DSSS Modulation will be used, with binary phase-shift keying (BPSK). Whatever the bit rate which a station is transmitting at, it will always use the same chipping rate. Thus, all signals will have the same occupied bandwidth, and processing gain will be inversely proportional to bit rate. Effective radiated power will be in direct proportion to bit rate, so that energy per bit is constant for all stations.

Different spreading codes correspond different "channels" of communications. Each station will have an assigned "hailing code," to which it will always be listening. When station A wishes to transmit to station B, station A will transmit using B's hailing code. In this first packet, A will send a code pair, one for a to use when talking to B, one for the reverse link.

In order to provide more reliable communications, with lower power, and higher user capacity, forward error correction (FEC) will be used extensively. The most likely candidate is Convolutional coding and Viterbi decoding. ASICs are commonly available now which are capable of high data rates with rate 1/2, and 1/3 codes and constraint lengths of 7 or 9. Other options might include combining convolutional codes with Reed-Solomon codes, or even using turbo codes. Again, these are all issues for the ground stations, and so could be changed without touching the transponder. Multiple schemes could be used concurrently, allowing experimentation to coexist with normal use.

2.3.2 Automatic Power Control

Automatic power control (APC) is necessary to make this system work. Without it, stations closer to the satellite would swamp out the ones further away.

APC guarantees that all signals will be received at the same strength, maximizing the number of them that can be decoded successfully. The pilot signal will be used as the reference power level. When a station is transmitting, it must constantly adjust its power up or down to make its downlink signal equal in power to the pilot signal. The actual downlink power received will vary, but the relative levels of the many signals and the pilot signal will remain the same.

2.3.3 Space Segment

The space module, the SSWBT itself, is a simple linear transponder, with only one addition. A simple (and small) circularly polarized patch antenna receives the many uplink signals at 2.4 GHz. After being amplified and filtered, they are downconverted to IF. At IF, the signal passes through a channel-bandwidth filter, and an AGC amplifier. Then a pilot signal is injected, and the combined signal is upconverted to 5.7 GHz. After it is amplified (about 25W output), it is retransmitted back to earth via another circularly polarized patch antenna.

The pilot signal is very crucial to the operation of the system as a whole. It allows the ground stations to have a reference power so that they are able to provide near perfect power control. It also provides a signal timing and doppler reference which the ground stations can also use to ease the problem of getting code and data synchronization.

2.3.4 Ground Segment

A minimal ground station, capable of transmitting digital voice, will be the typical end-user system. Such a station will use circularly polarized patch antennas, just like the satellite. It must have at least three despreading channels. One to monitor the pilot signal, one to monitor the station's own transmitted power and timing, and one for useful reception of signals from other stations. Since all of the signal processing associated with despreading channels will be done in digital logic in FPGAs or ASICs, adding more will not be difficult. Additional channels will be useful for receiving many datastreams at once.

3 Simulation

The SSWBT/ground station system was extensively simulated using the GOSSIP[5] simulation environment. GOSSIP is free software, available under the terms of the GPL, which allows simulation of complex systems. Primitive building blocks are designed using a C++ API. These blocks are tied together into more complex components, and entire simulation and postprocessing systems are created using GUILE, an implementation of Scheme.

3.1 Implementation

Simulation was performed on a chip by chip basis, at baseband, as a full RF simulation would have been prohibitively slow. Nonetheless, a full model of the radio channel was used, so accuracy should not be compromised. Traffic was modeled by random bit generators, as were the spreading codes of the stations. Channel noise is modeled as additive white Gaussian noise (AWGN), which is a good match for line of sight satellite paths. Worst-case (in terms of distance) paths are assumed in all cases.

3.2 Limitations

Certain assumptions were made in order to simplify the simulation process. FEC was not simulated.

The receiver was assumed to have perfect synchronization with the transmitter. Doppler shift was ignored. The effects of non-ideal amplifiers and filters were ignored, as these have not yet been designed. Despreading was simulated with correlators implemented in floating point, rather than the 2-, or 3-bit digital correlators likely to be used in an actual system. Automatic power control is assumed to be perfect, so that all stations are received at the transponder (and thus the ground station) at the exact same power. Most importantly, only 10 kilobit per second (digital voice class) stations were simulated, not the higher rate stations.

3.3 Design Parameters

There are many design parameters of the system which need to be selected. The aim of this simulation was to aid this selection, and thus many different designs were simulated. The main variables for the simulation were:

- Uplink and Downlink frequencies (2.4, 5.7 and 10GHz)
- Transponder Bandwidth (5 to 10 MHz)
- Transponder Model (Linear and Saturating)
- Transponder Power (25W and 50W)

Antenna gain of 6 dB on both ends of the link, and a total system noise figure of 2 dB on both ends of the link were assumed.

4 Simulation Results

For digital voice communications at 10 kbps, a bit error rate (BER) of 10-4 was chosen as the maximum tolerable. This is somewhat arbitrary, but is in line with figures often used in cellular phone systems. Forward error correction would be used in the system, but was not simulated, and correlation of BER with and without FEC is not always straightforward. Nevertheless, we can use a simple convolutional code for estimation purposes. The industry standard rate 1/3, constraint length 7, soft decision, viterbi decoder can obtain a 10-4 BER in the same conditions in which uncoded BPSK would achieve a BER of 0.02. As any FEC system which would actually be used would be at least as good as this code, we can assume that a raw BER of 0.02 or better would indicate acceptable performance.

Besides final BER, the bit error rate for a hypothetical receiver at the transponder input is also computed (shown as "Up" in the figures). This allows us to see the relative performance of the up and down links.

A number of interesting phenomena were observed in the simulation results. Normally in SS multiple access systems, as the number of stations increases, so does the BER due to mutual interference. While that effect is present, it is overshadowed by the power-sharing of the transponder. For example, when only one user is active, downlink power is shared by that one station's signal and by the noise input at the transponder. When there is a large number of active users, however, that station now gets a much smaller fraction of the total downlink power. If there are 100 users, each will get less than 1/100th of the power. Thus, the system is strongly downlink power limited, not congestion limited.

Normally, in SS systems, wider bandwidths allow for more users through higher processing gain, but have no effect on perfomance vs. random noise. However, when running through a transponder, wider bandwidths can actually hurt, because more noise will be received and retransmitted.

From the figure 1, one can see that up to 35 interferers, 5MHz wide spreading provides better results. It is not until there are about 60 interferers that 10MHz wide spreading provides a clear advantage. Unfortunately, neither system is very usable much past 60 users, at least under the assumption of the simple FEC code. Turbo codes could likely take the 10MHz spreading well beyond that.

The 50 W, saturated power transponder is clearly inferior to the 25 W linear one (see figure 2), despite the increase in power. This can be attributed to the lack of "soft" decisions in the satellite, which usually results in the equivalent of more than a 2 dB loss. It may have fared better if even wider spreading was allowed, but that is not really practical given the current amateur space band plans.

A "reverse" band plan was also simulated, with 5.7Ghz uplink and 2.4 GHz downlink. As can be seen in figure 3, this suffers from inferior performance. The signals arriving at the transponder already have poor BER, even before retransmission.

5 Conclusions

The SSWBT will open up a whole new world of digital communications to the amateur radio community. By taking advantage of underutilized spectrum, and advanced communications techinques, we will finally be able to interconnect the ham world with a high bitrate, integrated network. This will open up the possibility of digital videoconferencing, digital voice communications, and high speed data transfer. The simulations have shown that very practical digital voice communications are possible with the SSWBT, for over a hundred users at a time.

The simulations performed show that a bandwidth of around 10 MHz, with a linear transponder and uplink on 2.4 GHz provides the best compromise in terms of overall BER. The system can support almost 50 simultaneous speakers, or over 100 simultaneous conversations when configured like this.

The simulation will be refined to include the effects of doppler shift, imperfect power control, higher bit rate (higher power) stations, and more realistic component models. Separate simulations will be performed to model acquisition and maintenance of synchronization. Further simulation will be performed with various forms of FEC (convolutional coding/Viterbi decoding, concatenated with Reed-Solomon codes, and possibly Turbo codes).

References

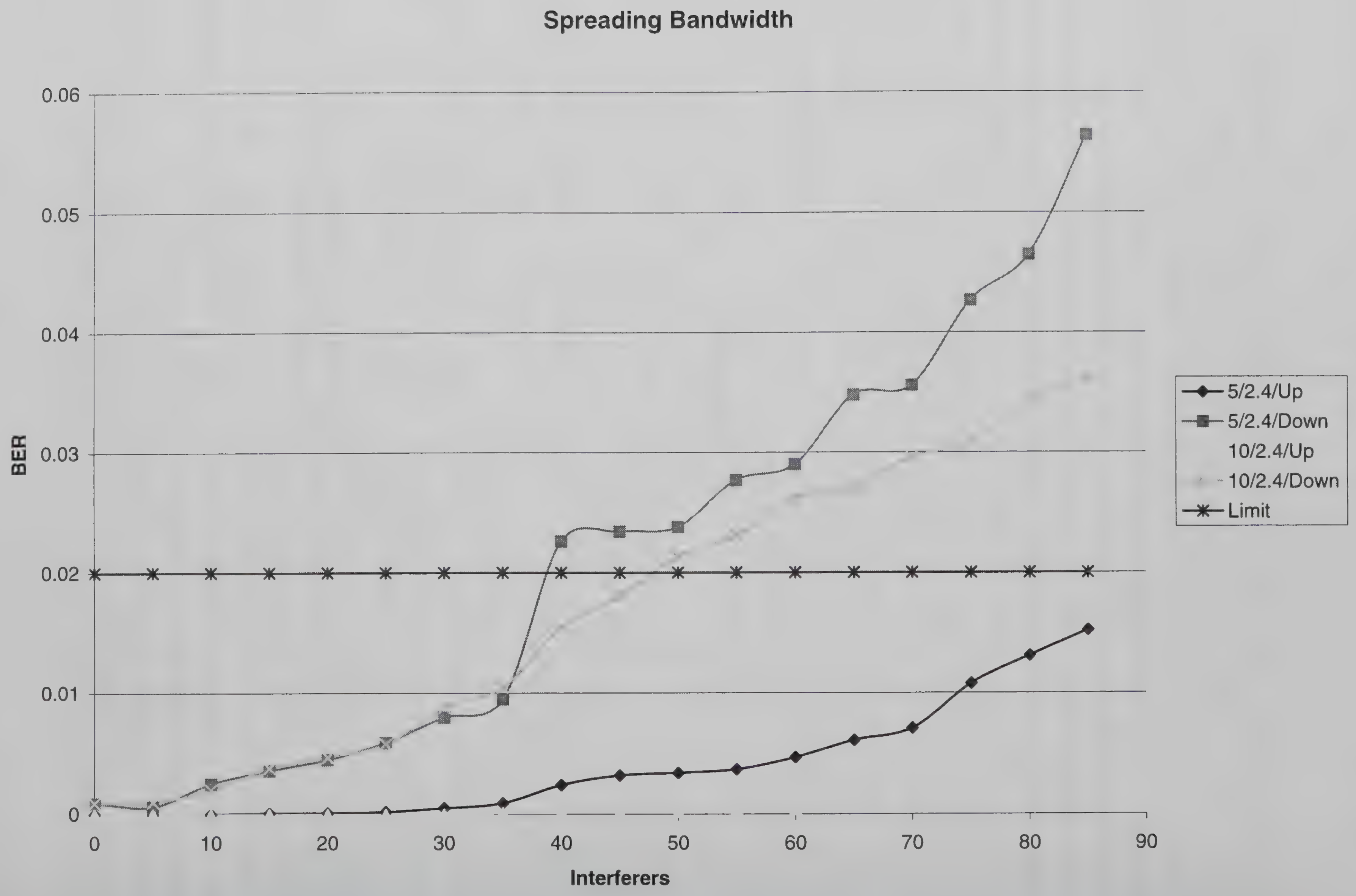
[1] Matthew Ettus. "Proposal for a Spread Spectrum Transponder Payload on the International Space Station." *Proceedings of the 18th ARRL and TAPR Digital Communications Conference*. Phoenix, Arizona. September 1999, pp 25-29.

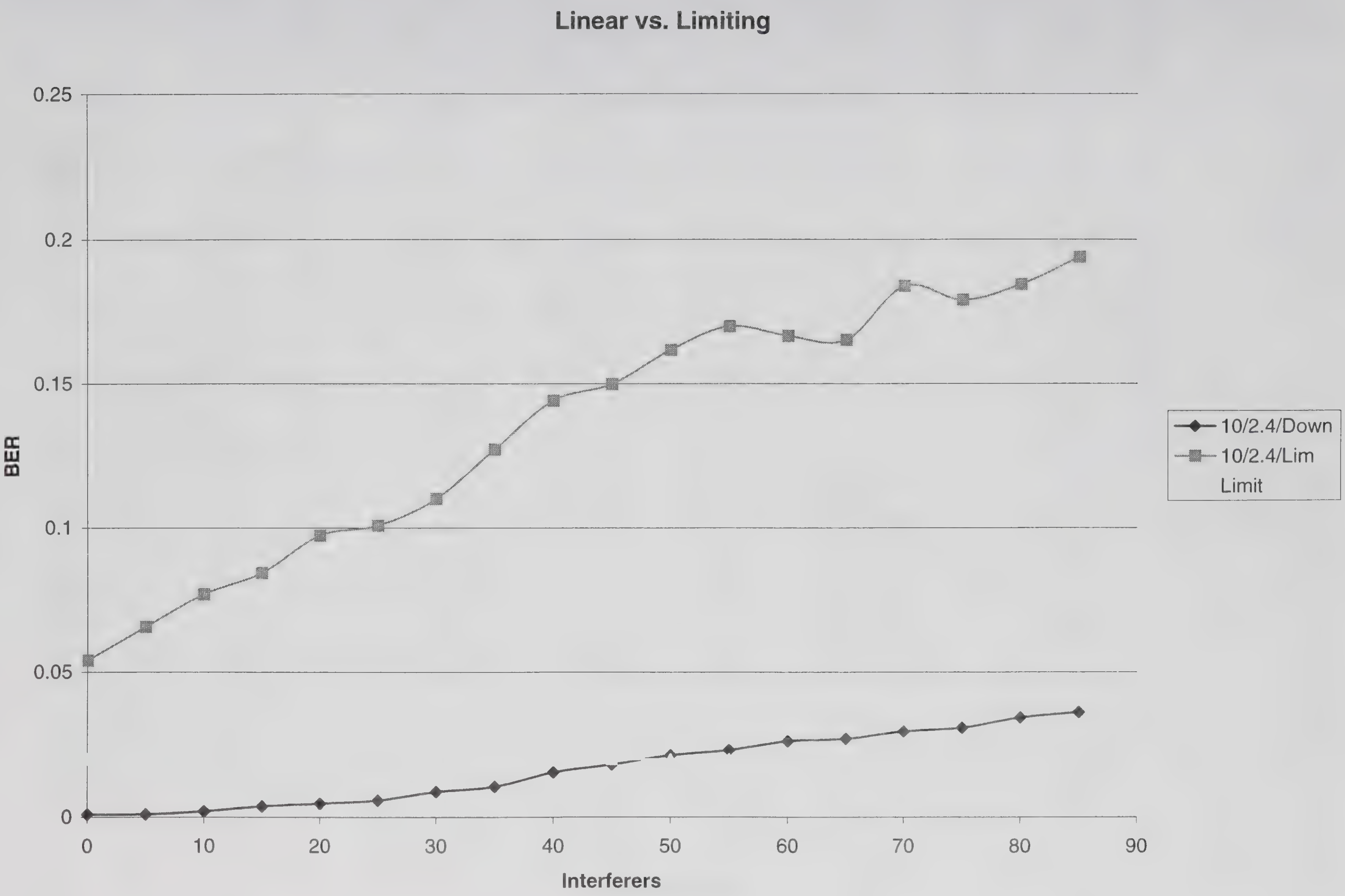
[2] Tom McDermott. *Wireless Digital Communications: Design and Theory*. Tucson Amateur Packet Radio. Arizona, 1996.

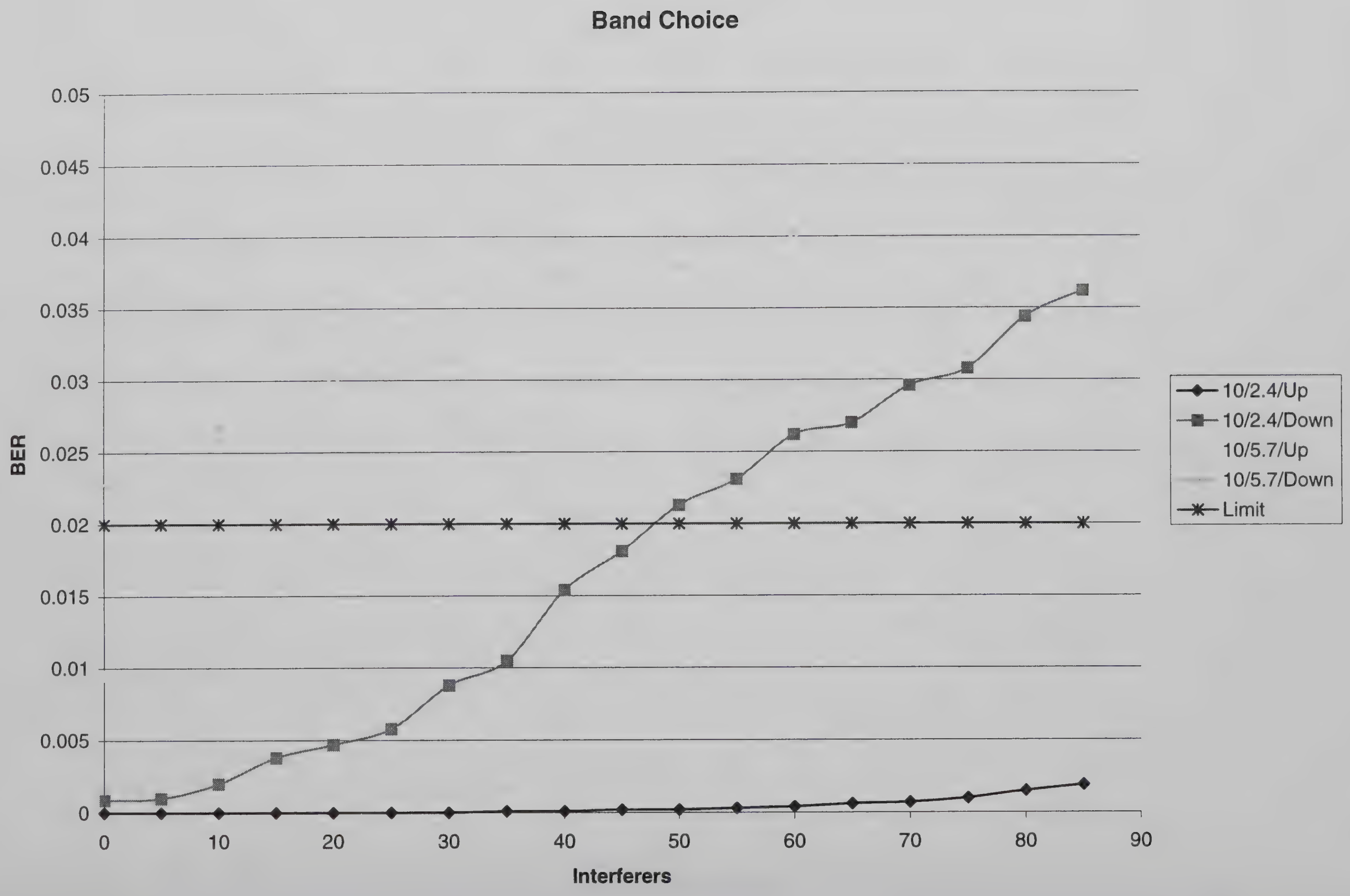
[3] *Qualcomm Forward Error Correction Data Book*. 80-24128-1B. Qualcomm, Inc. April 2000

[4] Theodore .S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall. New Jersey, 1996.

[5] GOSSIP Project Home Page. http://gossip.sourceforge.net


Spreading Bandwidth
BER
0.06
0.05
0.04
0.03
0.02
0.01
0
Interferers
0
10
20
30
40
50
60
70
80
90
5/2.4/Up
5/2.4/Down
10/2.4/Up
10/2.4/Down
Limit


Linear vs. Limiting
BER
0.25
0.2
0.15
0.1
0.05
0
0
10
20
30
40
50
60
70
80
90
Interferers
10/2.4/Down
10/2.4/Lim
Limit


Band Choice
BER
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
0
10
20
30
40
50
60
70
80
90
Interferers
10/2.4/Up
10/2.4/Down
10/5.7/Up
10/5.7/Down
Limit

Correlation for Direct Sequence Spread Spectrum–The design of a simple data extraction circuit, using serial acquisition in conjunction with a Delay–Lock tracking loop.

Panagiotis Gavalas.

Correspondence: Panagiotis Gavalas. e-mail: kirki@ath.forthnet.gr
A .pdf file of this paper can be retrieved at: http://wireless.net/PanosDSSS

Keywords: Direct Sequence Spread Spectrum, Correlation, Acquisition, Delay-Lock Loop, De-spreading.

Introduction

Spread Spectrum systems have become very 'trendy toys' in the areas of radio engineers and enthusiasts. Not only they provide privacy to the user, as an extend the ingenious channel encoding techniques involved make such systems very challenging for communications engineers and radio enthusiasts. Direct Sequence systems are the most preferred as they are the easiest to play with.

Sadly, among the articles and textbooks written on Spread Spectrum there are not many examples provided to the amateur radio enthusiasts. The concepts behind SS applications tend to be rather difficult to grasp and thus put the radio enthusiast in a very difficult position in terms of understanding and designing of such a system.

As an extend, some already available examples demand the use of complicated circuitry and expensive ICs. Thus, amateur radio enthusiasts are put in the risk of loosing valuable time and money in case of failure or possible damage of expensive components due to the idiosyncrasies of such ICs. Thus an easy and cheap solution is necessary.

This article is written in order to explain the concept of correlation in the simplest possible way to the amateur radio enthusiast and thus provide a simple solution concerning the heart of the system, i.e. the circuit that performs the de-spreading. It could be used as reference or a manual and the circuit described has the potential to be adjusted to any Direct Sequence SS system used for digital data transmition.

Demodulation and de-spreading for DS SS systems

At this early stage, the best way to understand correlation strictly speaking in terms of DS is to visualise the DS SS signal as a digital data stream with two types of modulation imposed onto it. At the transmitting side the following two actions are necessary.

- Spreading modulation, which 'spreads' the digital signal. It is used for channel coding purposes.
- RF modulation, which is the conventional part of modulation used for transmition.

Consequently at the receiving side of the system the exact opposite actions will have to be taken, in order to recover the data.

- RF demodulation, which 'cleans up' the spread signal from the radio frequency used for transmition. What is left after this action is the digital data stream 'carrying' the spreading code onto it.
- Demodulation of the spreading code. This is by far the trickiest part of the task and it is the ultimate recovery stage of the receiving system. It gives back the spread or 'hidden' data stream for the next stages of digital signal processing.

This article is solely concerned with the second stage of demodulation described above for DS SS systems. It is assumed that a spreading code cycle is used for each data bit.

This is exactly where the concept of **correlation** jumps in. The receiver must generate a replica of the exact same code used at the transmitter and modulo-2 add it (which is an equivalent mathematical operation to the XOR logic operation) with the received code signal. At this stage the received code signal is the code itself, either inverted or not corresponding to data bits '1' or '0' being received respectively. This is shown below, in Figure 1. [1]

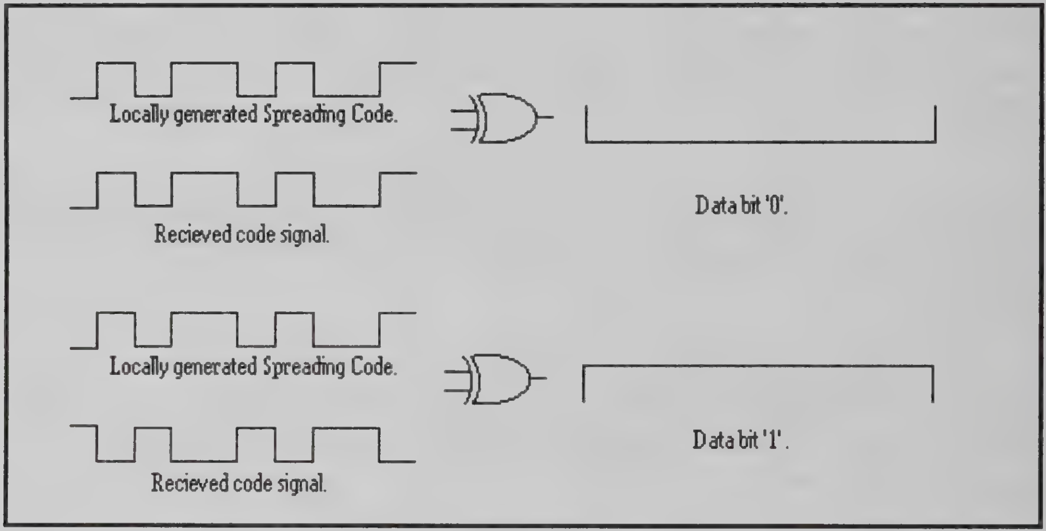


Figure 1 - Data De-Spreading in DS SS

Correlation versus time offset effects

A definition for correlation in DS SS should be given initially: "Correlation is the fundamental process for DS SS systems and is the action of measuring the similarity of the locally generated code, with the received code signal." The aim of this action as explained above is the completion of the digital data recovery.

In the previous paragraph it was assumed that the locally generated code and the received code signal were perfectly time-aligned. This case slightly departs from what the engineer will face in reality.
Due to uncertainties in the distance between the transmitter and the receiver, which may vary depending on how far the user may want to transmit or how far can the system transmit, propagation delays are caused. In addition to this, relative clock instabilities between the transmitter and the receiver spreading code generators will result into phase differences between the locally generated code and the received

code signal. Note that it is assumed that the receiver and the transmitter spreading code generators are not synchronised with eachother.

Thus, it is almost a certainty that when the signal will be received the locally generated code and the received code signal will not be in perfect time alignment. This is illustrated below in Figure 2. At a random time instant the locally generated code and the received code signal are modulo-2 added, i.e. XORed. Data cannot be recovered, as the two correlated signals are not time aligned.

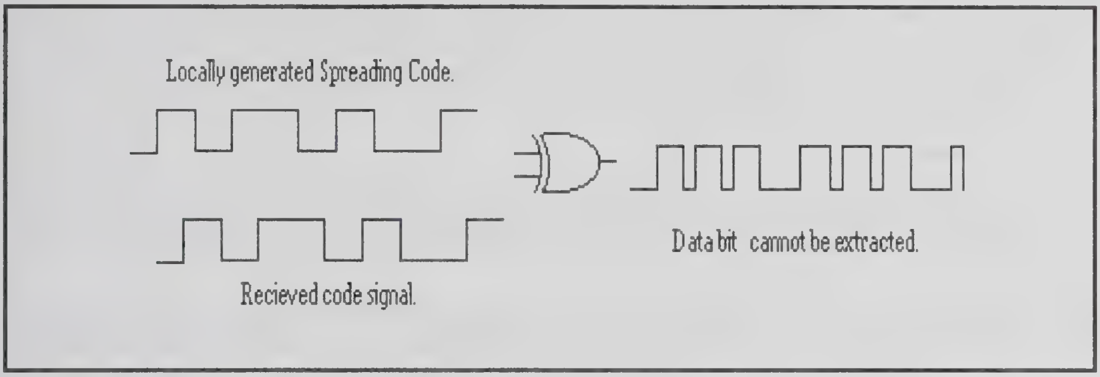

Locally generated Spreading Code.
Data bit cannot be extracted.
Recieved code signal.

Figure 2 - Despreading without the locally generated code being time aligned with the received code signal.

Mathematically, considering the locally generated code as a time varying function *f(t)* and the received code signal as *g(t)*, correlation is defined with the integral below:

$$\psi(t) = \int_{-\infty}^{+\infty} f(t)g(t)dt$$

Consider the case were a data bit '0' was transmitted. This effectively means that a non-inverted code cycle was received. The receiver will now have to time align the received code with the locally generated code. Thus mathematically the received code signal can now be re-expressed as a time shifted version of the spreading code. Therefore: $g(t)=f(t-\tau)$, and the correlation integral can be expressed as:

$$\psi(t) = \int_{-\infty}^{+\infty} f(t)f(t-\tau)dt$$

where τ is the time shift

This is the auto-correlation integral. Due to the properties of a well behaved maximal length code like the one that was used for the circuit described later, the correlation integral gives the triangle function as presented below in Figure 3.

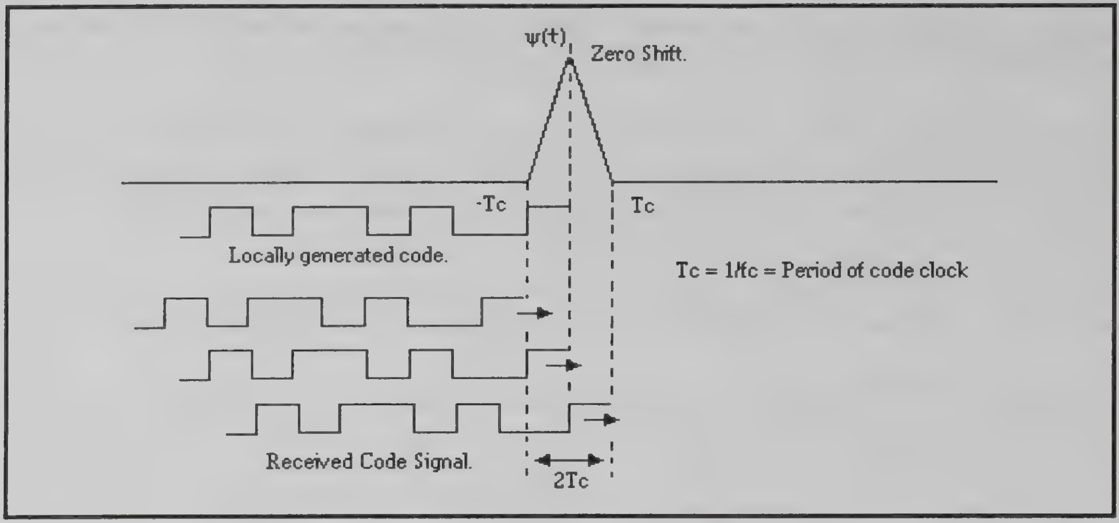


Figure 3 - The autocorrelation function of a maximal length code

The generation of this resulting triangle can be visualised by imagining the received code signal sliding past the locally generated code in search for perfect time alignment. In terms of mathematics the integral simply symbolises the area generated under the product of the received code signal and the locally generated code as one slides past the other. Due to the properties of maximal length codes any product out of the region where the two correlated signals are one chip apart will be negligible compared with the products within the time region of one chip separation. Maximum output occurs for zero time shift, i.e. for $\tau=0$, or otherwise perfect time alignment.

The use of maximal length codes is strongly recommended as their correlation integral produces a single triangle making de-spreading an easier task.

If a data bit '1' was transmitted then the case is quite similar except for the fact that the correlation triangle would appear with 'pointing' downwards. [1]

Acquisition

This section deals with the ideas and considerations of how to bring the locally generated code and the received code signal into perfect time-alignment. This process is known as acquisition.

The device designed to perform acquisition must be able to produce a voltage output proportional to the similarity between the two correlated signals. This voltage then, will have to be compared with a threshold, set to statistically decide whether the two signals are perfectly time aligned or not. If they are not perfectly time aligned, the acquisition circuit must then decide to perform a systematic search through the time and phase uncertainty region between the two signals and thus find the exact time instant when the two signals are perfectly time aligned.

Acquisition can be performed serially or in parallel. The cheapest solution is provided by a serial search, which is the technique used for the circuit that will be described later. [1]

The frequency uncertainty problem

Suppose that perfect time-alignment, between the locally generated code and the received code signal has been accomplished. One could say it is obvious that data could be extracted from that point. This is not the real case.

Due to minimal, but not negligible relative frequency offsets between the receiver's clock and the transmitter's clock the received code signal will arrive at a slight different frequency compared to the locally generated code. As a consequence, when synchronisation has been achieved the two signals will try to drift apart from each other resulting loss of synchronisation. Again, it is assumed that the receiver's and the transmitter's code generators are not synchronised with respect to each other.

This is another problem that the designer must overcome. It is of major importance to maintain synchronisation. The idea is to force the receiver's code clock to run at the same frequency as the received code signal. This is achieved by means of feedback loops and the process is called tracking.

Tracking is used in conjunction with acquisition must be initiated right after synchronisation has been achieved. [1]

Tracking methods

The two methods used are the delay lock loop and the tau dither loop. The delay lock loop is presented in Figure 4 below. The received code signal is split into three different channels and is mixed with three replicas of the spreading code. Two of the replicas have the same time offset with respect to the third. One is advanced by a fraction of one chip time duration and the other is retarded by the same fraction of one chip time duration.

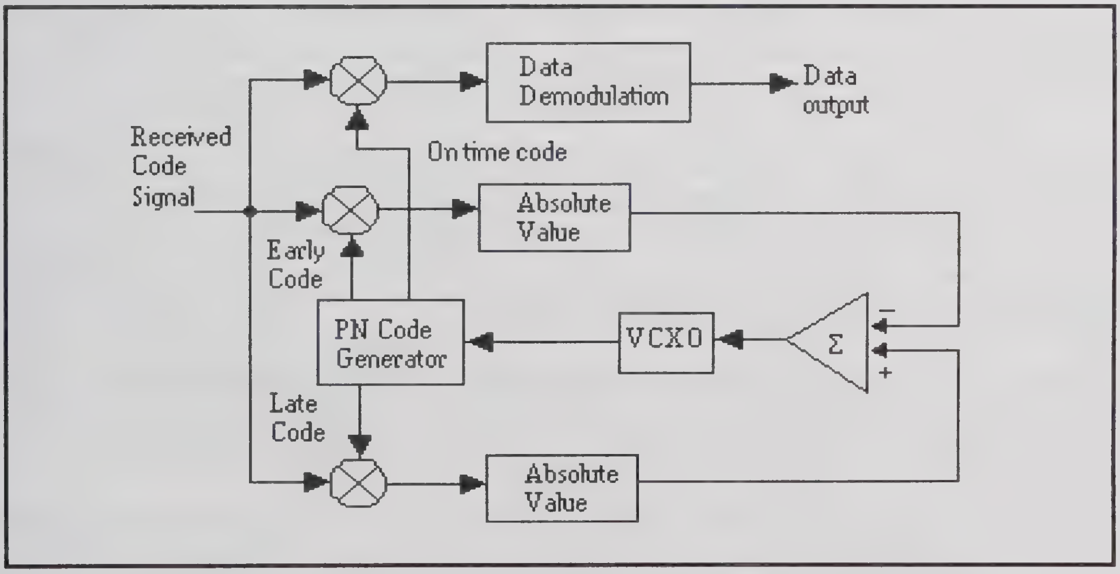


Figure 4 - The Delay Lock loop for DS spread spectrum signals

These two replicas are used for the delay lock loop whereas the third is used at a separate channel to perform acquisition. Thus the names 'early' and 'late', with respect to the 'on-time' code. Due to the fact that both of the codes have the same time offset with respect to the on-time channel, when perfect time alignment is achieved at the on-time channel, their mixing stage outputs are made equal with the aid of absolute value circuits independent of whether a data bit '1' or '0' was recovered. Thus solely the degree of matching is converted to DC levels.

As the received code signal and the locally generated code tend to drift apart from each other, the absolute value outputs will differ. Their difference is used to steer the voltage control crystal oscillator (VCXO) that drives the code generator. Thus the locally generated code can be forced to catch up in frequency with the received code signal. Consequently perfect time alignment between the locally generated code and the received code signal is maintained and data can be continuously extracted.

The tau dither loop does not provide the engineer with the same flexibility in terms of steering as the delay lock loop and will not be examined. [1]

The de-spreading circuit's block diagram

This circuit was designed for a DS SS system with a digital data stream at 9600 kbps. A maximal length code of 255 chips was used.

The received code signal is split into three channels, one performing serial sliding acquisition and data extraction, while the other two are used for the delay lock loop as presented in Figure 5.

In an earlier part of this report the de-spreading concept was explained with the aid of XOR gates for simplicity. Double balanced mixers substitute the XOR gates for this application. The double balanced mixers are configured in such a way to give output voltage amplitudes corresponding to the degree of matching between the received code signal and the locally generated code.

It is recommended that, the two code replicas used for tracking, should be separated by a half chip with respect to the on-time code. This will be explained in the next paragraph. For this particular application the codes were stored in an EPROM (Erasable Programmable Read Only Memory).

Initially the VCXO is switched to a certain input tuning voltage and hence the locally generated code is clocked at a smaller frequency compared to the clock frequency of the transmitter. This effectively provides serial sliding acquisition. Thus the locally generated code will 'see' the received code signal slide past it, at a relatively higher frequency. Thus, a scan through the total time and phase uncertainty region is achieved until the two signals are perfectly time aligned.

At the acquisition channel, two outputs were taken from the DBM. One is used for data extraction whereas the other is used to indicate, whether synchronisation has been achieved or not and thus turn the delay lock loop on.

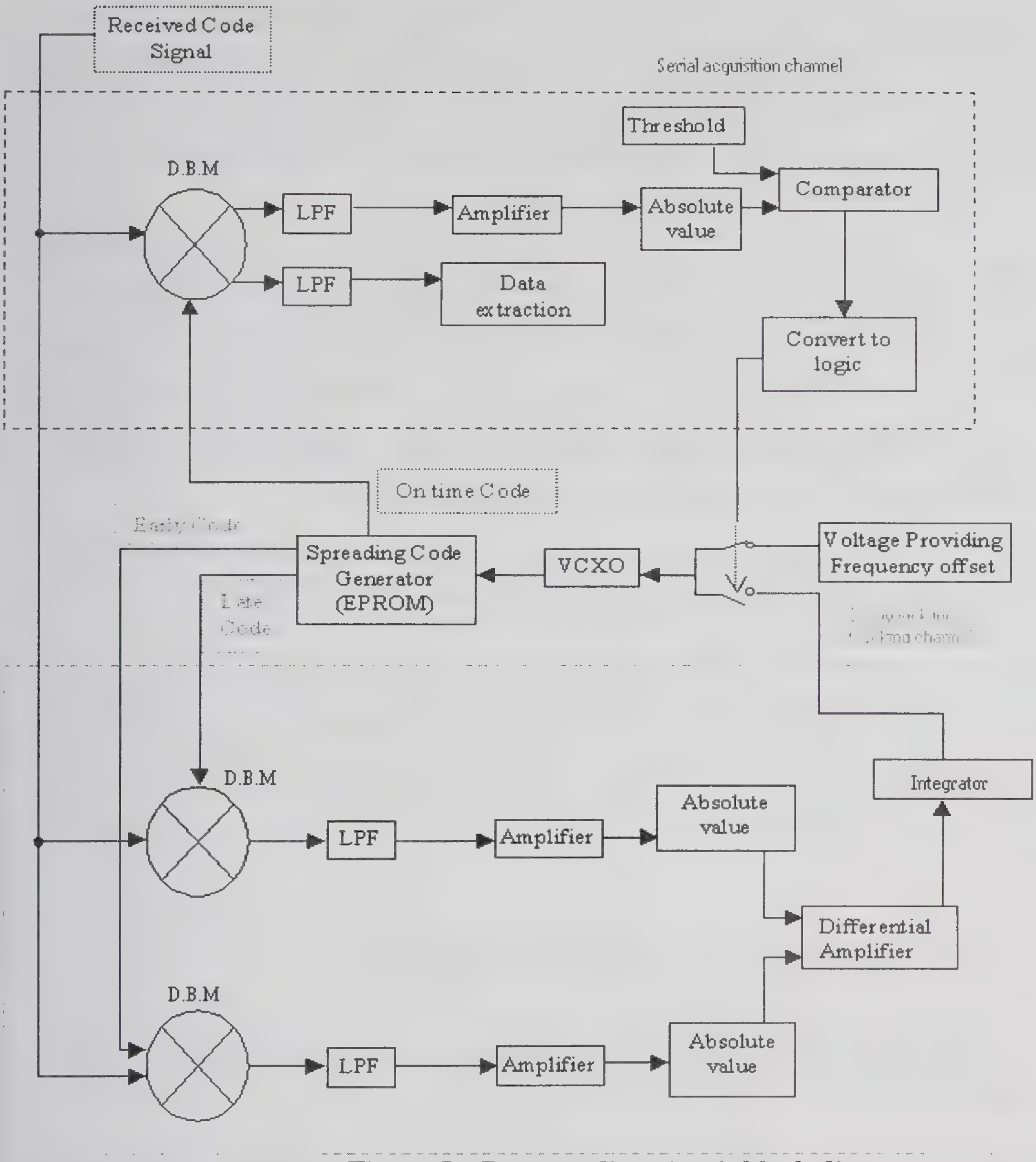


Figure 5 – De-spreading circuit block diagram

Both of the outputs of the DBM are low pass filtered to get rid of the unwanted noise delivered to the de-spreading circuit together with the received code signal after the RF demodulation stage. The second output after low pass filtering is amplified in order to provide larger voltage levels corresponding to the degree of matching. Here, the aim is to provide a dc indication corresponding to how well the two signals are time aligned with respect to each other.

The output of the DBM used to provide an indication whether synchronisation has been achieved or not, gives its highest possible output voltage value when the locally generated code is perfectly time aligned with the received code signal being the code inverted corresponding to a data bit '1' transmitted. The lowest possible voltage value is output when the locally generated code is synchronised with the received code signal being the code non-inverted corresponding to data bits '0' transmitted.

Consequently two different voltage levels correspond to the same degree of matching between the locally generated code and the received code signal. The circuit has to be able to identify both of these cases as the same in order to provide an indication whether synchronisation has been achieved or not, independent of the bit that was initially transmitted. Thus an absolute value circuit is used, and for both of the cases described above the same voltage levels were output, corresponding to the same degree of matching, independent of the transmitted bit.

This voltage level is then compared with a threshold voltage in order to provide an indication whether synchronisation has been achieved or not. At the time instant where synchronisation was achieved the threshold was exceeded and the output of the comparator switched to is maximum. The output of the comparator is then converted to logic levels and hence when synchronisation is achieved the tuning voltage of the VCXO, is switched from the one that was providing the frequency offset resulting to sliding acquisition, to the output of the delay lock loop. The VCXO is tuned to run at the same frequency of the transmitters code generator frequency when no error signal is delivered from the delay lock loop. By this switching operation the delay lock loop's operation is initiated.

The exact same tactics are used at the other two channels, the early and late channel, in terms of mixing the two signals and the conversion of the degree of matching to voltage levels. Thus low pass filtering, amplification and absolute value circuits are used for the reasons explained above.

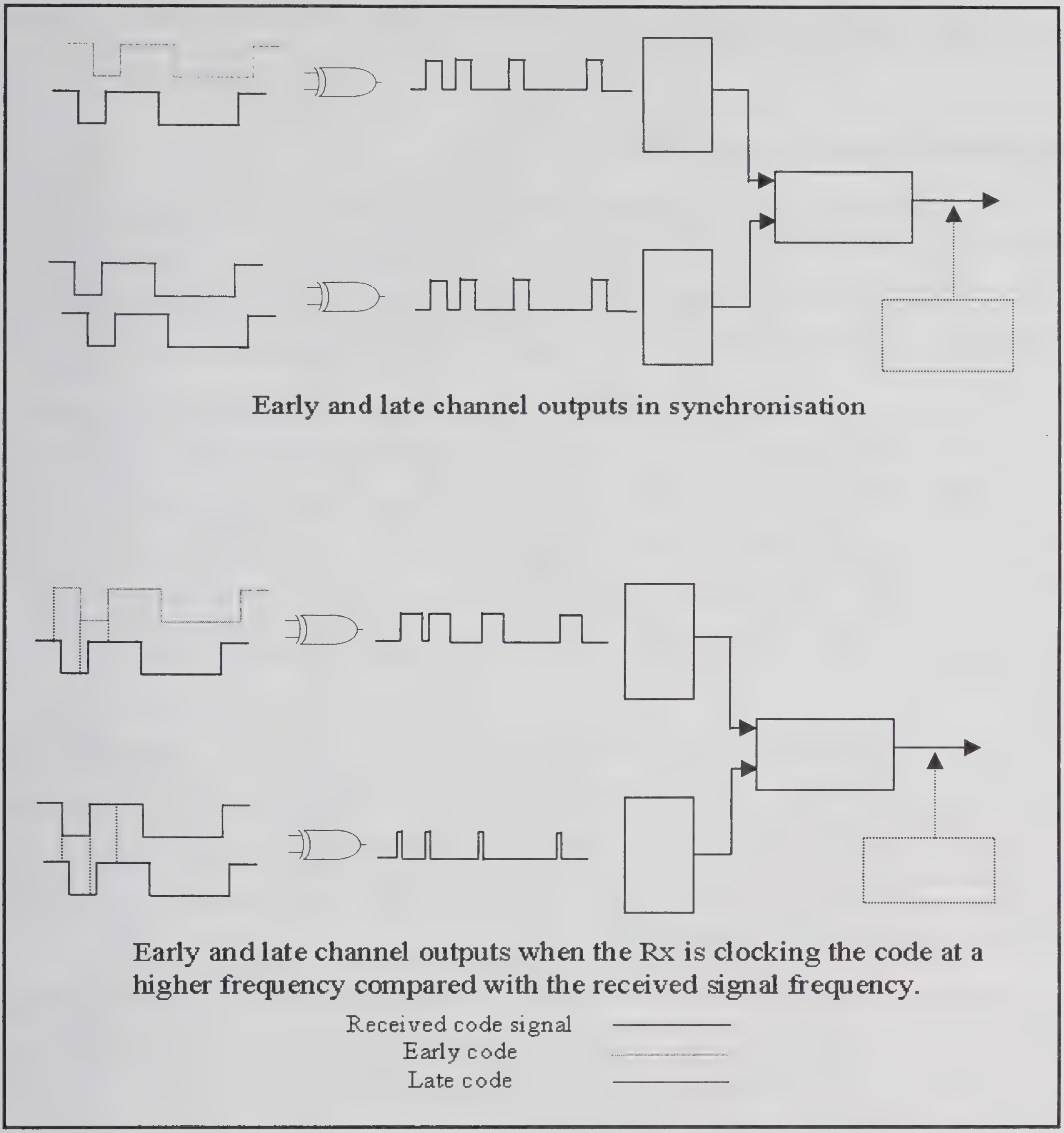
To maintain synchronisation the outputs of the absolute value circuits are summed in a difference amplifier and their difference generates an error voltage with the aid of an integrator to steer the VCXO, forcing it to catch up with the received code's frequency. The integrator's output is proportional to the integral of the output of the difference amplifier, resulting frequency changes to the VCXO proportional to the rate the two signals are drifting apart from each other. Thus continuous data extraction is achieved. [1]

Half chip separation of the early and late with respect to the on time code

This part presents a simple explanation why a half chip separation should be preferred for the late and early spreading codes with respect to the on time code. Note that here the XOR de-spreading concept is employed again. Figure 6.

Suppose that the received code signal and the locally generated code are synchronised. At this certain time instant due to the half chip separation, the outputs of the XOR gates at the early and late channel are exactly the same. If they are both integrated, then the outputs of the integrators will be voltage levels proportional to the areas under the outputs of the XOR gates. (i.e. the integral of the XOR outputs.)

Supposing that at this time instant the locally generated code is driven by the VCXO at a higher frequency compared to the frequency of the received code. The outputs of the XOR gates will differ as presented in Figure 6.



Early and late channel outputs in synchronisation

Early and late channel outputs when the Rx is clocking the code at a higher frequency compared with the received signal frequency.

Figure 6 – The effect of a half chip separation between the early and late codes with respect to the on time code as the two signals drift apart from each other.

As a result the integrators will deliver different voltages at their outputs, as the area under the outputs of the XOR gates is different.

But it is the rate of change between the outputs of the integrators, which causes the VCXO to catch up with the received code signal's frequency. The smaller the time distance between the early and late codes with respect to the on-time code the larger the rate of change at the output of the difference amplifier. [1]

Detailed analysis of the de-spreading circuit

MC1496 active DBMs were used. An alternative version of the product detector (refer to MC1496 datasheet) circuit was used, and is presented in Figure 7. Both of the on-time channel DBM outputs were used (U1). The 47kΩ pot's wiper must be centered to eliminate the presence of the spreading code on top of the data stream, when the two signals are synchronised. At pin 6, when the locally generated code was synchronised with the received code non-inverted the output was at 6.4V. For synchronisation with the received code being inverted the output was at 9.8V. Pin 12 was the complement of pin 6.

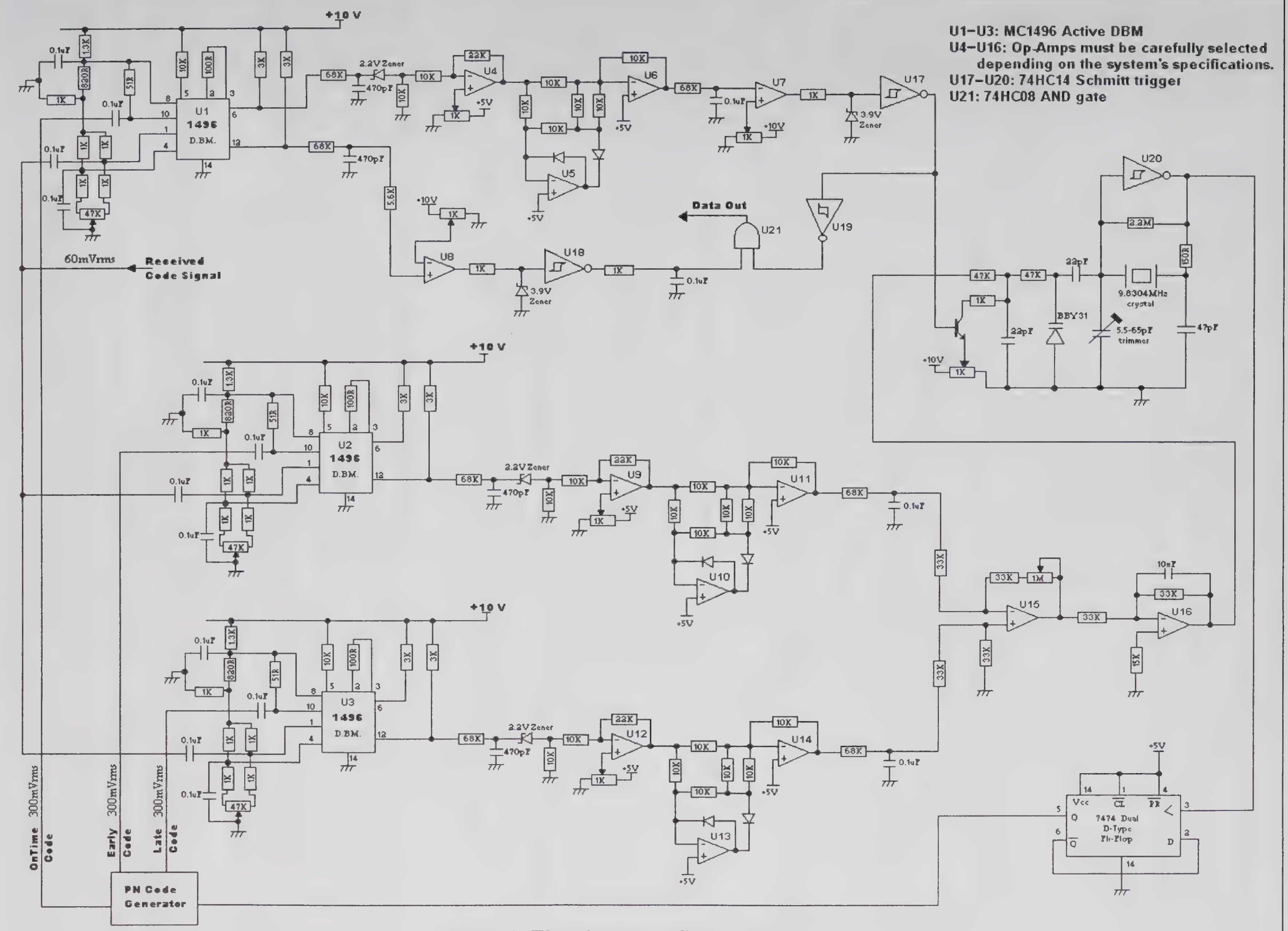
Both of the outputs where then low pass filtered by first order RC filters. The output from pin 6 was then scaled by a 2.2V zener diode down from 4.2V up to 7.6V for synchronisation achieved between the locally generated code and a non-inverted or inverted received code respectively. After low pass filtering an amplifier was used (U4) with one input referred to ground and the second to a voltage divider giving not only the ability to amplify the signal but also to set a 5V mid-line between the maximum and minimum outputs which were from 2V up to 8V. This trick gave the ability to the full wave rectifier (U5&U6, i.e. the absolute value circuit) to output voltages corresponding solely to the degree of matching between the two signals. Thus the output of the rectifier was made independent, whether a data bits '0' or '1' where de-spread. The output of the rectifier was from 5V to 8V.

After low pass filtering again the rectified signal was compared with a fixed threshold voltage at approximately 6.5V halfway between 5V and 8V (U7). Due to the nature of the code used for this system the degree of matching is always minimal for every shift position except for the region were the correlation triangle is generated. This is mainly the reason why a well-behaved maximal length code should be used. Thus the voltage amplitude at the output of the rectifier was always very small (about 5.1V) and the largest possible (about 8V) was rapidly output when the two signals where perfectly time aligned. Thus a threshold decision at 6.5V was a good one. The output of the comparator (U7) was from 1.5V to 8.5V, indicating synchronisation at 8.5V.

This output was then scaled down from 1.5V to 4.6V by a regulator, which is effectively a resistor connected with a 3.9V zener diode to ground. These voltage levels were then converted to logic with Schmitt trigger inverters twice. The first Schmitt trigger inverter (U17) gave a high output while the circuit was trying to acquire the signal and a low output when synchronisation was achieved. The other Schmitt trigger inverter (U19) gave the opposite voltage levels under the same conditions.

Pin 12 output of the DBM was used for data extraction. After low pass filtering again the output was compared with a fixed voltage set at 8.1V, right at the middle between the minimum and maximum output voltages. The output of the comparator (U8) was scaled down and regulated as above, converted to logic using a Schmitt trigger inverter (U18) and low pass filtered.

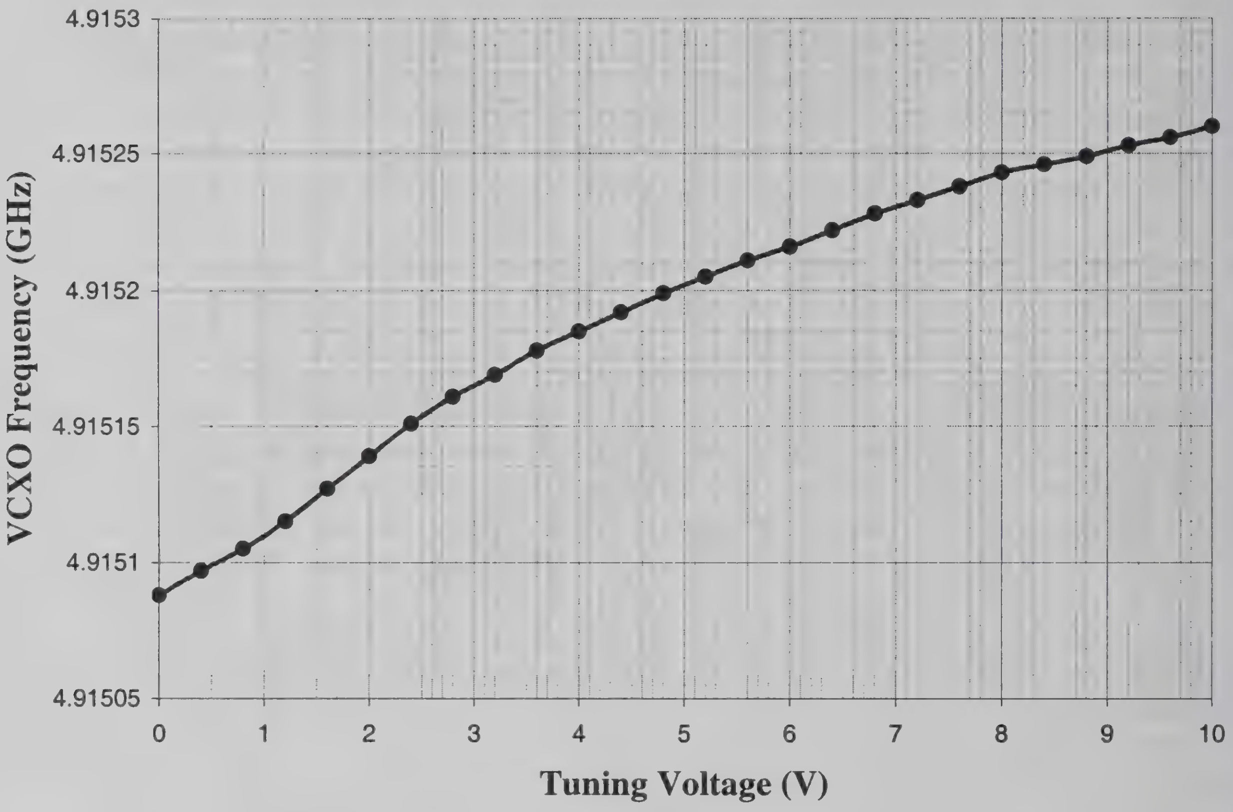
This inverter's (U18) output, was effectively the data when the locally generated code was synchronised with the received code. The output of the second inverter (U19) described above was then input to an AND gate (U21) together with the data. By this action the data was passed to the audio side of the system only when synchronisation was achieved.



U1–U3: MC1496 Active DBM
U4–U16: Op-Amps must be carefully selected depending on the system's specifications.
U17–U20: 74HC14 Schmitt trigger
U21: 74HC08 AND gate

Figure 7-The de-spreading citcuit

For this particular application a 9.8304MHz crystal was used at theVCXO. The frequency of the VCXO was divided by two using a D-type flip-flop. By slowly adjusting the trimmer capacitor, the output of the flip-flop was tuned at 4.9152 MHz, which was the transmitter's code generator frequency, for 5V input tuning voltage. A 5V voltage corresponds to no error voltage delivered to the VCXO by the delay lock loop. See Graph 1.



Graph 1 - The VCXO frequency vs. the tuning voltage (after the /2 stage)

The VCXO was initially given a slight frequency offset providing the sliding of the received code against the receiver's locally generated code. This was achieved by pulling the tuning voltage of the VCXO down to 2.4V by using the first inverter's output (U17) to turn the transistor on. This frequency offset, of about 50Hz, provided the ability to the circuit to slide a complete cycle of the received code against the locally generated code at approximately 4 seconds. Thus the maximum time, to run a complete scan through the total time uncertainty region time, perfectly time align the two signals and thus extract the data was about 4 seconds.

When synchronisation was achieved, the inverter's output was turned down to 0V and thus the transistor switched off. Hence the VCXO tuning voltage was then fed by the delay lock loop. This is effectively how the circuit switched from acquisition to tracking mode.

The same DBMs amplifiers and rectifiers were used for the early and late channels. Both of the outputs of the full wave rectifiers of the late and early channel are passed into a difference amplifier (U16) and an integrator (U18). At the difference amplifier a 1MΩ variable resistor in series with a 33KΩ resistor was used in order to vary the amplifier's gain. Consequently, the rate of change of the error voltage was made variable. Thus the VCXO was forced to catch up with the received code signal's frequency at a faster rate and stability of the delay lock loop was improved.

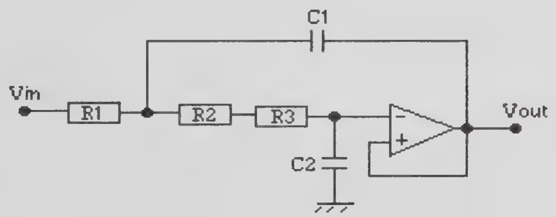
This could be used as an example. Any crystal could be used and the frequency offset can be easily adjusted by playing with the trimmer capacitor. Filtering may also change depending on the application. [1]

Results and recommendations

The circuit was able to re-construct the data. Exhaustive tests where carried out with the circuit's early late and on time channels constructed on printed circuit board while the VCXO and the code generator were still on breadboards. Initially the circuit did manage to lock and extract the data even with 60dBs of attenuation. In order to test the circuit's ability to recover data under the worst signal conditions the attenuation was set to exceed 60dB and the LNB was set not facing the transmitter. The circuit did not manage to extract data under those conditions.

In reality, for a DS spread spectrum system that is to be tested in environmental conditions, this would turn out to be a problem that must be overcome.

In terms of circuitry the first and obvious thing to do is to use improved low pass filtering at the outputs of the DBMs. A filter with a sharper 3dB frequency is required. Thus it would be wiser to use an active low pass filter of a higher order. Active low pass filters use op-amps and thus the designer should take the frequency response of the op-amp itself into account. As a solution a 'Sallen and Key' second order low pass filter is shown below with a 3dB frequency of around 4.650kHz (Corresponding to 9.7kbps). It is assumed that the 3dB frequency is in the frequency range of the op-amp. Higher order filters can be used, providing sharper 3dB frequencies but a second order filter should be enough. The capacitor's and resistor values should be selected depending on the application.



$$f_{3dB} = \frac{1}{2\pi\sqrt{R1(R2+R3)C1C2}}$$

R1=R2=68k, R3=1K, C1=C2=470pf

Figure 8 - Proposed 'Sallen & Key' second order active low pass filter

As a second priority, a smaller acquisition time must be provided. The acquisition time of the circuit presented in Figure 7 was about 4 seconds. This is a problem when the user is trying to align the receiver with the transmitter for ling of sight transmition. Thus rapid acquisition is essential. In order to achieve this, a larger frequency offset should be provided. But first it must be ensured that the frequency offset is within the delay lock loop's capture range. For the circuit configuration presented in Figure 7, the 50Hz frequency offset was the maximum within the loop's capture range and it was worked out using a trial and error approach.

By substituting the 22pF capacitor between the 470kΩ resistor and the 9.8304MHz crystal, with a larger capacitor the VCXO's frequency deviation for a range of tuning voltages from 0V to 10V will be larger. For the 22pF capacitor the frequency deviation was about 175Hz. It was found that for a 33pF capacitor the deviation was 225Hz and for a 56pF capacitor 260Hz. For every capacitor used it was ensured that a tuning voltage of 5V corresponded to about 4.9152MHz. (i.e. the transmitter code generator's frequency). This was accomplished by adjusting the 5.5-65pF trimmer capacitor.

In order to keep the desired frequency offset within the loop's capture range, the time constant of the 1kΩ resistor and 22pF capacitor RC circuit, connected from the collector of the transistor to ground, must be made smaller. By reducing the time constant of the RC circuit the VCXO will switch from its acquisition mode to tracking mode faster. Thus by trial and error again the right resistor and capacitor values can be worked out and consequently the acquisition time can be minimised.

With the current circuit configuration, when the transmitted signal is corrupted, or lost for a short time period, the circuit will switch back to acquisition mode from tracking mode until the data is de-spread again. Extra circuitry could be added in order to keep the circuit on tracking mode for a short time period, before switching back to acquisition mode when the signal is lost. Thus if the lost signal appears again within this time period the circuit will not have to try and de-spread the data again. Such an idea will involve complicated circuitry and it may not be necessary if the acquisition time was made quite small (i.e. a fraction of a second). When the signal is lost, the circuit may switch back to acquisition mode, but when the signal appears at the receiver again the acquisition time will be quite small and data will be extracted again rapidly. [1]

References

[1] GAVALAS P. - MAY C. M. A. - PROCTOR, M. D. - WINKLE, J. L.: **'The design of a simple Spread Spectrum system at 10GHz'**, (The University of Sheffield UK, Department of Electronic and Electrical Engineering, 2000)

Recommended Reading

1 SKLAR B.: **'Digital Communications: Fundamentals and Applications'**, (Prentice Hall, 1988), pp. 561-570
Comments: A great introduction.

2 DIXON, R. C.: **'Spread Spectrum Systems'**, (Wiley-Interscience, June 1984), pp. 120-260
Comments: This is a 'must read' book on Spread Spectrum.

3 **'The ARRL Spread Spectrum Sourcebook'**, (American Radio Relay League Inc, 1997), Chap. 8
Comments: This is the kind of book I was praying to get my hands on. It helped me a great deal. It covers everything one needs to know on Spread Spectrum. Practical solutions and examples are provided.

4 AGARD (Advisory Group for Aerospace Research and Development): **'AGARD lecture Series No. 58 on Spread Spectrum Communications'**, (NATO, 1973), Chaps. 4-5
Comments: This is a tough one. All aspects are explained in the greatest possible theoretical detail. Be very patient while reading this one and take notes.

5 DIXON, R. C.: **'Spread Spectrum Techniques'**, (IEEE Press, 1976), Part V
Comments: A good theoretical reference. Presents articles written on Spread Spectrum.

6 COOPER R. G., McGILLEN C. D.: **'Modern Communications and Spread Spectrum'**, (McGraw-Hill, 1986), pp. 296-356
Comments: Another 'must read' book.

7 MAZDA F. F.: **'Electronics Engineer's reference book'**, (Butterworths, 5th edition), pp. '39/12-14'
Comments: Good solutions for circuit design and construction. All the considerations that can be easily forgotten are there.

An MFSK Mode for HF DX

Murray Greenman ZL1BPU
94 Sim Rd, Karaka,
RD1 Papakura,
New Zealand
Coombedn@ihug.co.nz

Nino Porcino IZ8BLY
Via dei Tulipani 21,
89133 Reggio Calabria,
Italy
Ninopo@tin.it

Abstract

Development of a very robust and sensitive modem for Amateur DX use. A description of the main problems involved in receiving digital modes on HF, and ways to counter them; a brief history of MFSK, and a description of MFSK technology. A description of the design process of a new MFSK mode for amateurs, its performance, and the software used to demonstrate it.

Concept

Multi-tone FSK modes have never until now been used in Amateur radio circles, and it is hard to understand why. Examples of these modes do exist, for example some of the older robust radio-teletype modes used for fixed links in the diplomatic service.

The author was dissatisfied with the DX performance of existing HF rag-chew modes, such as RTTY, PSK31 and MT63, especially on long path, and decided to tackle the solution to reliable digital DX communications by first considering the problems. An analysis of these prompted an investigation of slow baud rate multi-tone FSK to counter multi-path problems, while retaining immunity to Doppler flutter.

An appreciation of the performance of historical robust modes, especially Piccolo, led to the idea of replicating something similar in a modern environment, making the most of the DSP forces now available. The idea was of course to achieve and confirm the performance promised in theory, and claimed in the papers of the time. Using PC based DSP software allowed many different signaling parameters to be tested, and the best combinations chosen. The results of the tests led directly to the development of two new modes for Radio Amateurs, equal to others in sensitivity, and superior in handling poor propagation conditions. MFSK16 and MFSK8 have been designed as conversation modes, suitable also for nets and broadcasts.

The Problems

There are three main problems that make HF digital communications, and especially long path DX, a real challenge:

1. Multi-path propagation, which causes fading, selective fading, and timing changes.
2. Polar flutter, which modulates the signal, especially affecting the phase.
3. Noise, QRM, carrier interference, man-made crud and lightning noise.

1. Multi-path

Multi-path affects digital transmissions in several ways. Perhaps the most difficult to handle is the different time-of-flight of signals arriving at the receiver over different paths. As the signals change in strength, the timing can change slowly, or suddenly shift by 5 – 10 ms, and sometimes more. Often two rays of similar strength and different timing will exist. If the signal is arriving long-path, and short path is also viable, this difference can be as much as 50ms. Obviously data symbols as short as 32 ms (at 31.25 baud) are seriously affected – as successive symbols are differently timed or even run over the top of each other. These multi-path problems can be overcome to a large extent by using lower baud rates. At 8 baud for example, the symbols are 128ms long.

By "windowing" the data in the time domain before performing an FFT, the beginning and end of each of the symbols are ignored. Of course it is at the transitions between symbols where the timing problem is most apparent, and thus multi-path affects can be further rejected.

Multi-path reception also causes fading of the signal, as the rays from different paths cancel. If this happens across the bandwidth of the signal, the best strategy is to use as much sensitivity as is possible. If however the cancellation is frequency selective, which will be the case with short differential delays, the transmission must be recovered with some of its components severely attenuated or missing. 160 meters is perhaps the most difficult band in this regard, with slow and very deep selective fades. The best strategy for this problem is frequency and time diversity, for example multiple tones, with an FEC technique and interleaver to fill in the missing data.

Multi-path can really only be conquered using very low baud rates.

2. Polar Flutter

Flutter or "Doppler" is a much misunderstood phenomenon, and has until recently been largely ignored in Amateur circles. The arrival of PSK31 on the scene brought this problem to prominence. When signals pass through the ionosphere, especially in the region of the poles, they are subjected to quite large variations in refractive index, and therefore velocity. These changes in the reflective and transmissive properties of the ionosphere vary all the time, and can vary very quickly in a random way, resulting in both frequency and phase shift of the signal. At the receiver, the signal appears to change in amplitude as well due to multiple ray cancellation and augmentation.

Very narrow-band signals are especially prone to flutter problems, as the carrier is essentially Doppler modulated by the ionosphere. With low baud rates the problem is exacerbated, as the carrier phase has more opportunity to change during each symbol than at higher speeds. It is for this reason that differential PSK is used. PSK31 operates right at the limit for differential PSK modes on HF, and as digi-DX operators have found, PSK31 is of limited use on polar paths, both short path and long path. Even worse, QPSK31, which offers promise due to added FEC, is in fact worse than PSK31 because the phase modulation per symbol is reduced from 180° to 90°, and the phase error margin is similarly reduced. From experience, QPSK31 is really only of use on VHF.

Polar frequency flutter also occurs. However, provided a non-coherent detection system is used (not phase sensitive), polar flutter has little effect on FSK modes with reasonable shifts. Flutter effects of the order of about 5 Hz peak-peak are typical. Doppler frequency modulation broadens the received signal, so the receive detector must cope with this effect. MFSK uses an orthogonal detector capable of rejecting this spillover energy. Traditional RTTY achieved this rejection using wide spaced tones.

3. Noise and QRM

Coping with noise (white noise) is a matter of sensitivity. PSK modes have been demonstrated to achieve very high sensitivity, but then so can MFSK modes, as has been shown with MFSK16 and MFSK8.

QRM, such as SSB splatter, Pactor interference and so on, requires an error reduction system to control and limit the damage. Much the same applies to lightning noise, although this tends to have very high energy and lower occurrence rate. Burst noise such as this is best countered by a strong FEC system with a good Interleaver.

Rejecting crud such as power noise is to some extent a sensitivity issue, and partly a bandwidth issue. The advantage of MFSK in this regard is that the bandwidth of each individual receiver is very narrow and the diversity is good. Good detector decision strategies recover the most likely data despite high levels of noise.

Carrier interference (TV birdies, Morse, AM broadcast stations, PSK31) is a problem with most modes. Narrow modes avoid this better by taking up less bandwidth, so reducing the risk. Very wide modes such as MT63 achieve resistance to carrier interference through redundancy and diversity. MFSK resists carrier interference well, unless the carrier is strong. If this is the case, the carrier captures the detector decision system and reception stops. Intelligent decision systems can effectively notch out a constant carrier, and through the use of FEC good copy returns.

The History

Having reviewed the problems to be solved, it became obvious that what was required was a low baud rate, fast, time and frequency diverse, phase-insensitive system with narrow bandwidth! These may sound like ambitious and contradictory requirements, but one class of transmission does have these properties – MFSK, Multiple Frequency Shift Keying. Very little information is readily available about MFSK, and virtually none in Amateur literature, since it has never before been used in Amateur service.

There have been a number of commercial and military applications of MFSK, the most notable and groundbreaking example being Piccolo, developed in the 1960's by the Diplomatic Wireless Service for British Foreign Office diplomatic fixed link service. Most MFSK systems have now fallen into disuse, to be replaced by faster satellite links and ALE systems.

Perhaps the earliest MFSK system for HF was the LMT "Seven Tone Radio Printer", developed in France in 1935.[1] This non-synchronous system used seven wide spaced concurrent tones, and a direct printing system not dissimilar to Hellschreiber.[2] Such a system would today be called Multi-Tone Hellschreiber. Another system, rarely used on HF, is the familiar DTMF telephone tone system, which has a two-of-eight tone set.

Coquelet was developed by ACEC in Belgium for police and military use during the early 1960s. This equipment was relatively portable, largely electro-mechanical but not very sophisticated, using reeds for the tone generators and filters. An important paper published by ACEC demonstrates that the error rate on a digital radio channel is lowest when the number of symbols per character is the least possible.[3] Coquelet operated asynchronously used high and low tone groups, 12 tones, and two symbols per character.

Piccolo was developed to provide a secure and reliable link between diplomatic posts. The first permanent link was established between England and Delhi in 1965, and later with Singapore, using a 32-tone system. Later systems used 6 or 12 tones. Piccolo was significantly more sophisticated than other systems of the time, and used a synchronous technique, two symbols per tone, and amplitude modulation to provide symbol framing.

The most important invention used in Piccolo was the integrate-and-dump detector, with loss-less filters quenched at the start of each symbol period. In one stroke this device provided very high sensitivity, a robust receiver, and most important of all, very good rejection of adjacent channel energy. This detector provided orthogonal detection with spacing equal to

[1] "Seven Tone Radio Printer", *Electrical Communication, 1937* by L. Devaux and F. Smets.

[2] See http://www.qsl.net/zl1bpu for the history and modern use of Hellschreiber systems.

[3] "Les Téléimprimeurs, Téléchiffreurs et Transcodeurs ACEC - Système Coquelet",
ACEC-Revue, No 3-4, 1970.

the baud rate – 20 Hz spacing at 20 baud. An excellent analysis of the detector[4] in the time domain shows how the energy from one tone cancels in the detector for the adjacent tones if the signal is sampled at a baud rate equal to the channel spacing. A modern analysis in the frequency domain[5] comes to the same conclusions.

What is MFSK?

MFSK, Multi-Frequency Shift Keying or M-ary FSK, is a technique where one or more tones bursts from a set of many tones are transmitted at one time. These events are the smallest entity of the transmission, the "symbol". Unlike RTTY (which is 2-ary FSK), each MFSK symbol can carry more than one data bit. In fact, the number of bits depends on the number of tones used. Most MFSK systems transmit just one tone at a time,[6] and so:

$$N = log_2m$$

(where N = number of bits/symbol and m = number of tones in symbol set)

It can be very confusing to realize that the transmission data rate exceeds the baud rate, but that is most certainly the case. Not only does this give MFSK a needed advantage, since the baud rates need to be low to counter multi-path effects. but by using error reduction with care, the frequency diversity provided by an MFSK system adds worthwhile robustness.

Fig. 1 A single MFSK tone burst

MFSK symbols are traditionally hard-keyed, in other words, the symbols are sharp edged carrier tone bursts as in Fig. 1, giving the signal a certain harshness and perhaps unnecessary bandwidth. Of course these tones are not transmitted in an isolated manner as shown above, but are a continuous train of slightly differing frequencies. The signal quality and bandwidth can be improved considerably through the use of phase continuous tone generation (as has been done for years in RTTY). With a good sine wave phase continuous modulator the bandwidth of the transmission will be very little more than

$$Bandwidth = 2.m + 2.b$$

(where m = number of tones in symbol set, b = baud rate)

The following diagram shows the characteristic sin(x)/x shape of a hard keyed tone burst. The first nulls are either side of center by the baud rate, and each successive null is spaced by the baud rate multiplied by an integer.

[4] "Multi-tone signalling system employing quenched resonators for use on noisy radio- teleprinter circuits", *Proc. IEE* Vol 110 No. 9, September 1963, H.K. Robin OBE et al.

[5] "Wireless Digital Communications: Design and Theory", Chapter 4, Tom McDermott N5EG

[6] 2-of m systems do exist. DTMF of course is one example – G3PPT's Throb is another.

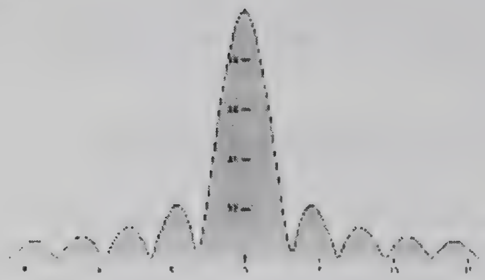


Fig. 2 Frequency domain response of a single tone burst

The big advantage of a hard keyed system is that the transmitter need not be linear. There is no amplitude modulation and no raised cosine envelope to control the switching noise or transfer symbol clock information, so no inter-modulation can occur. The Piccolo system did transmit a symbol pair clock, by keying the screen grid of the class C transmitter to change the power level by 10%. Coquelet achieved symbol sync by using unique symbol sets for the first and second tone of each pair. Our new system is single-symbol oriented and uses only the symbol transition properties to provide clock recovery. There is no amplitude modulation.

Theory (both earlier time domain analysis and recent frequency domain analysis) tells us that orthogonal reception of close spaced tones with an asynchronous detection system can only be achieved when the tone spacing equals the symbol rate (baud rate) or multiples thereof:

Tone spacing for orthogonal reception = b.i (where i is an integer)

Piccolo was probably the first mode to achieve a spacing of i = 1; Coquelet used mechanical reeds as filters and used i = 3. With FFT techniques i = 1 is easily achieved.

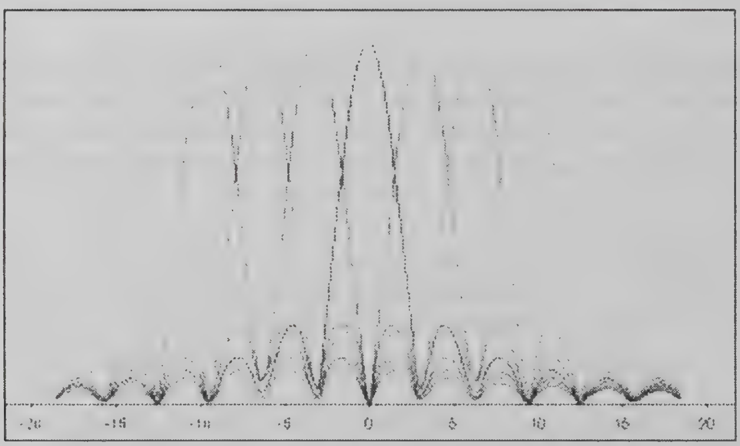


Fig. 3 Seven MFSK tones spaced at i = 1

In Fig. 3 it is easy to see how the frequency domain null of one tone fits exactly under the peak of the next, when the correct spacing is chosen. The diagram also gives some idea of the overall spectrum of an MFSK transmission.

In the receiver, a separate detector follows each tone filter, and the weight of each filter is compared to find the one with the most energy at the end of the symbol period. The integrate-and-dump filters do not respond to random noise, due to the integrative nature of the technique, and thus the system is very sensitive. Further, since the system is orthogonal, each filter responds only to the matched tone, and not its neighbors. In fact a null occurs at the end of the symbol period when the neighbor tone is received (which is why the integer spacing is used).

All these techniques (except phase synchronous tones) were used in Piccolo. There are however a number of modern digital techniques which can be added to an MFSK system to further enhance the performance. Here are just a few:

- Modulator weighting using Gray code for minimum Hamming distance and therefore minimum bit error.
- Multiple baud rate options using tone spacing = baud rate.
- FEC with auto-synchronising Interleaver to gain error reduction and rejection of systematic interference such as carriers. The Interleaver adds time diversity as well as easing the load on the FEC decoder. Unpredictable order of coded dibits recovered by using trial decoders.
- Symbol shape correlation or carrier phase related symbol phase recovery.
- Synchronous FFT filter and demodulation, with a soft decision decoder. Decoder phase and amplitude metrics used for AFC, tuning display and signal / noise meter. Soft decisions fed to FEC decoder.
- Windowed FFT for excellent rejection of multi-path delays of 10ms or more.
- Highly efficient varicode alphabet, with extended ASCII character set, and super-ASCII control codes.
- The MFSK modem can be completely realized in DSP, using a modest Pentium class PC with a 16 bit sound card

Almost all of these techniques are now in use in the MFSK16 software already released. The addition of FEC and the Interleaver means that the signal can be decoded with little more than 50% of the available data clearly received. The use of soft decisions could add several dB in sensitivity (yet to be realized). The Varicode improves the text throughput by in excess of 25% compared with ASCII. The robustness is appreciable. The modem is inexpensive and simple to install and operate since only a PC with sound card is required.

Designing a New Mode

It was obvious from the research that the time had arrived to make a serious attempt at designing a new MFSK mode. While one of the Piccolo versions could have been replicated, it was thought best to start afresh, with no preconceived ideas. It might seem a tall order, but this was the design brief:

- A mode insensitive to multi-path, with a baud rate of 16 baud or less
- A mode insensitive to polar flutter, and robust in noise and QRM
- A typing speed in excess of 35 words per minute
- A constant amplitude transmission, with minimum bandwidth and constant phase
- Tone spacing equal to baud rate – transmission to fit within a standard CW filter
- Transmission to be bit-stream based, with single symbol per data block
- Varicode alphabet allowing extended ASCII character set and control codes
- Full-time strong convolutional code FEC with Interleaver.
- Orthogonal decoder using integrate-and-dump techniques
- The complete modem to be DSP, operating with a PC sound card

Plenty of useful feedback and good advice was received when the MFSK16 Specification was first issued, via the TAPR HFSIG and a number of other email groups frequented by DSP and coding experts. In addition, Nino Porcino IZ8BLY offered to graft the system into his “Stream” development software, designed for easy experimentation with different digital modes. This provided a very quick way to try out the various options. Nino is a DSP coding expert of no mean ability.

Baud Rate

The baud rates chosen, 15.625 and 7.8125 baud, are precise binary divisions of 8000 Hz, making sampling simple. It was decided to keep the baud rate and tone spacing fixed at the same numeric values, so at 15.625 baud the tone spacing is 15.625 Hz. With 16 tones, this gives a signal 316 Hz wide. With 7.8125 baud and 32 tones, the bandwidth is about the same.

At these baud rates, the channel data rate is the baud rate multiplied by the number of bits per symbol – four for 16 tones, and 5 for 32 tones. So, the channel data rate is 62.5 bps or 39.1 bps respectively, leaving room for an FEC system and still plenty of typing speed!

Transmitter Keying

In the interests of minimum bandwidth and maximum simplicity, a fixed amplitude transmitter modulation scheme was chosen, using a numerically controlled constant phase frequency shift keyed (CPFSK) sine wave generator to create the tones. In effect each symbol is a hard-keyed tone burst, but the transmission bandwidth is kept to a minimum by constant phase slewing from one tone to the next, with no bursts transmitted in isolation. Each symbol is of the same duration, and by virtue of the numerical relationship between baud rate and tone spacing, each symbol also starts and finishes at the same phase, although the complete number of cycles per symbol increases from one tone to the next. This can be important as an aid to recovering the symbol clock.

The major advantage of such a system is that transmitter non-linearity cannot cause the signal to become any broader. Of course the audio stages must remain linear to prevent audio second harmonic effects spaced up or down the band.[7]

Modulation Scheme

It was decided to weight the tones according to a gray-code, to minimize the bit errors when the signal is mistuned. The lowest frequency tone was chosen to represent binary "0000" or "00000" for 16 or 32 tones respectively.

The chosen modulation scheme was a synchronous, single symbol system. Therefore no symbol pair framing was necessary. Unlike Piccolo and other such modes, there was therefore no need to indicate the start of a symbol pair, and it was decided to attempt to recover symbol phase with no extra symbol synchronizing signal. (Remember, for example, that PSK31, also a symbol synchronous system, amplitude modulates the signal at the symbol rate to provide clock recovery). Several suitable methods of symbol clock recovery are known. The symbol clock can only be recovered if the data is varying – synchronism is lost if an idle carrier occurs.

The transmission consists of an apparently random bitstream, largely due to the effects of the Coder and Interleaver. When the transmitter falls idle (the buffer is empty), streams of zero bits are flushed through the Coder and Interleaver, resulting in an idle carrier at the lowest frequency (used for signal tuning). A null character is inserted occasionally during idle to keep the receiver symbol clock in alignment.

Receiver Demodulator

At the time the specification was issued in June 2000, several developers were experimenting with FFT detectors for digital mode demodulation, and it had been demonstrated by Lionel Sear G3PPT,[8] Pawel Jalocha SP9VRC[9] and others, that the FFT could be operated synchronously with the received symbols. For MFSK the FFT samples were windowed using a raised cosine envelope to exclude the first and last few milliseconds of the sample period, effectively providing a "guard band" to eliminate much of the multi-path problem.

The synchronous FFT accurately models the integrate-and-dump detector. It not only integrates the symbol throughout the whole symbol period, but it exhibits the same rejection in adjacent channels for orthogonal detection, and also matches the square transmitted pulse well.

[7] If the audio tones used are much below about 1.4 kHz, the second harmonic (created by audio clipping) will still be within the transmitter crystal filter passband.

[8] "Throb", a simple concurrent tone pair MFSK system working at one or two baud

[9] "MT63", a 64 tone PSK parallel carrier system with FFT phase demodulator

There are a number of areas in the receiver that could be improved. The decision decoder that takes the amplitude values from the FFT bins needs to have a better strategy than simply picking the strongest value. A strategy also needs to be developed to cope with carrier interference, which, if strong enough, can cripple an MFSK system. The decisions from the decoder should be soft, and the soft values applied to a soft decision FEC decoder.

The receiver has AFC, but currently the range is limited and it only operates on an idle carrier. There is room for development of a wide full-time AFC and also a strategy for receiving when mistuned by exactly one tone.

Alphabet

We chose a varicode alphabet similar to that developed for PSK31.[10] In fact the PSK31 alphabet was used at first, until it was realized that the idle synchronizing requirements were quite different to PSK31, and it was possible to assign many more short length codes than is possible with PSK31. This was achieved by defining the inter-character framing sequence as "001" rather than "00", where the "1" is the first bit of the following character. This allowed "000" and "0000" and so on to be valid sub-sequences within characters. With other minor changes, such as giving higher priority to numbers and backspace, the throughput on plain text (average 7.4 bits/character) is some 15 – 20% better than PSK31 at the same bit rate.

Forward Error Correction

Several FEC schemes were experimented with, but for the sake of simplicity and universal understanding we went with the NASA standard R = ½, K = 7 coding[11] and the excellent Viterbi decoder developed by Phil Karn KA9Q.[12]

For every input data bit there are two output data "dibits", generated by two polynomials from the taps of an 8 stage shift register. Call the outputs of these registers O_0 to O_7. The polynomials are each the modulo 2 sum (XOR) of five of these register outputs. The polynomials are:

$$\boldsymbol{Dibit_1 = O_1 + O_3 + O_4 + O_6 + O_7}$$
$$\boldsymbol{Dibit_2 = O_1 + O_2 + O_3 + O_4 + O_7}$$

The dibits defined above are multiplexed into a single data stream, in order Dibit_1 then Dibit_2.
Perhaps the most unusual aspect of FEC coding in the MFSK scenario is that the dibits are multiplexed into a single data stream, and no fixed relationship need exist between the coded dibits and the symbol weighting. The receiver therefore needs to work out for itself which order the dibits are in. This can be done with two Viterbi decoders, choosing the output with the best metrics. We elected to use a main decoder and a trial decoder sampling one bit apart, switching them around if the trial decoder gave better results. In MFSK16 the order of dibits is in fact constant and known (always the same weight in the symbol), so for lower performance computers a single decoder solution would be practical. This is not however true for MFSK8, since there are five bits per symbol. (Maybe an R = 1/5 code is called for!)

Interleaver

The all-important Interleaver proved at first to be a problem. It is very important with an MFSK system to provide a good interleaf, since each symbol carries four or five bits. Loss of one or two adjacent symbols can cause serious problems in a Viterbi decoder. Most Interleavers require a framing system such as a Costas Signal for synchronism, and it was felt that since simplicity was important, we should attempt to find a way to use bit position in the symbol to provide automatic

[10] "PSK31 Fundamentals" by Peter Martinez G3PLX, http://aintel.bi.ehu.es/psk31theory.html
[11] "The MFSK FEC Coder", Nino Porcino and Murray Greenman, http://www.qsl.net/zl1bpu/MFSK
[12] "Convolutional Decoders for Amateur Packet Radio", Phil Karn KA9Q, http://people.qualcomm.com/karn/papers/cnc_coding.html

synchronism. It actually turned out to be very easy – by using an interleaf table the same width as the number of bits per symbol, and reading the bits out diagonally, a modest but fully automatic Interleaver resulted.[13]

Imagine the data bits to be transmitted can be represented by

"ABCDEFGHIJKLMOPQRSTUVWXYZ0123456789....."

and so on. Then the bit stream about to be sent is arranged by the Interleaver into a table as deep as the number of bits per symbol, like this:

8 Tones:	16 Tones:	32 Tones:
ADGJMPSVY2...	**A**EIMQUY...	**A**FKPUZ...
B**E**HKNQTWZ3...	B**F**JNRVZ...	B**G**LQV1...
CF**I**LORUX14...	CG**K**OSW0...	CH**M**RW2...
	DHL**P**TX1...	DIN**S**X3...
		EJOT**Y**4...

The bits are arranged vertically as they arrive from the Coder, in a four bit wide FIFO shift register, but are sent on to the modulator ***diagonally***. After each diagonal is sent, the FIFO shifts left and the first column is discarded. The first group of bits sent has been highlighted in each case in the above table. So the bits sent are:

8 Tones: **AEI** DHL GKO JNR MQU PTX SW1 UZ4 ...
16 Tones: **AFKP** EJOT INSX MRW1 ...
32 Tones: **AGMSY** FLRX4 KQW39 ...

Once again the first group of bits sent on in each case is highlighted. The spreading of the bits is quite modest since the table is not very deep, but since the order of the bits is always known at the receiver (from the bit order in the received symbol), synchronization of the Interleaver is automatic. The receiver employs a similar FIFO register, writes received bits in vertically, and reads out the result on the opposite diagonal to the way they were sent.

To improve the bit spreading, ten of these simple interleavers are simply stacked one after another. The result is a spread of some 94 bits, with a 71 bit delay. The MFSK16 receiver will copy with up to four consecutive symbols knocked out.

Performance

The new modes were first of all tested on air under the conditions for which they were conceived – long path DX. IZ8BLY and ZL1BPU had been in at least weekly communication for some time, and during this phase contacts were stepped up to almost daily, using gray-line long path. Before this software came along, the most reliable communications were on 20m using PSK-Hell (also invented by ZL1BPU and coded by IZ8BLY), largely because it is very sensitive and not as badly affected by polar flutter as PSK31. Once the new MFSK modes came along, it was immediately obvious that the performance exceeded every other mode previously tried! To make conditions more difficult, and at the same time buying some slight privacy, tests were moved to 18 MHz, where signals were generally much weaker, while still suffering polar flutter and multi-path. Once again, copy was excellent, coping with fades into the noise without loss of copy. Receiver tuning took a little skill, but with reasonably stable transceivers and AFC, comfortable rag-chews for well over an hour became the norm, most of the time using as little as 5W. This is an 18,000 km polar path (22,000 km long path).

Tests on other bands soon followed, and it was quickly apparent that the new modes were also as good as any other on most HF paths, if a little more fussy in tuning than most. Users have been most impressed with the way that QSOs are viable under conditions that would not have been considered possible. On the lower bands, 160m to 40m, no other modes coped as well with the combination of lightning and multi-path reception. The legendary robustness of MFSK has been fully

[13] The IZ8BLY Diagonal Interleaver, Nino Porcino IZ8BLY, http://www.qsl.net/zl1bpu/MFSK/INTERLEAVER.doc

realized. 3,500 km night-time QSOs between ZL-VK and across Europe were relatively easily achieved with just a few watts. At no time is more power than 25W necessary. An additional bonus has been the reports of excellent VHF performance. European VHFers report that knife-edge diffraction and inversion fades are less of a problem than with PSK31, and the MFSK16 is not affected by aircraft doppler problems.

Transmitter adjustment is easy, and the signal is invariably clean and sharp to tune across. The musical not is not unpleasant on the ear, and users quickly recognise and learn to tune the different modes, although a transmission on the wrong sideband is hard to identify until the transmission is briefly idle.

Johan Forrer KC7WW has tested MFSK16 and many other modes using an ionospheric simulator. Subjectively, MFSK16 provides better copy under CCIR POOR conditions than any other rag-chew mode tested so far.[14] In AWGN tests by Moe Wheatley AE4JY, the sensitivity was equal to that of Digipan (PSK31). These results are achieved at very respectable typing speeds.

The two MFSK modes chosen for release are:

Mode	Modulation	Baud Rate	Channel bps	Bandwidth	Typing Speed
MFSK8	32-FSK	7.8125	39.0625	316 Hz	26 WPM
MFSK16	16-FSK	15.625	62.5	316 Hz	42 WPM

The modes were named for their approximate baud rate, rather than the number of tones. As can be seen from the above table, the bandwidth of the two modes is the same, and comfortably fit through a CW filter. The two can be easily differentiated by ear, as MFSK8 at 7.8 baud is clearly audible as a sequential and musical progression of tones, while MFSK16 sounds more like a jumble of sound. The typing speed is all that is necessary for a rag-chew mode. The turn-around time between overs is about two seconds. Included in this time is an idle period at the start of transmission, which allows the receiver AFC to reacquire the signal. At the end of the transmission the buffer is again flushed until the idle carrier is heard.

MFSK8 is a little more sensitive, and a little more robust than MFSK16, while the typing speed is still reasonable. The tuning accuracy required is rather tight, and the turn-around time doubled. It has been found from experience that these disadvantages are a small price to pay for the opportunity to complete a QSO as the band fades out, or to make a QSO on LF or MF where one would not otherwise have been possible! The turn-around time is still rather less than for MT63.

MFSK16 and MFSK8 seem to have very good immunity to most forms of noise and interference. Testers have deliberately tried transmitting MT63 and other modes on top of MFSK16, and found to their amazement that neither mode was disrupted by the other! The sole remaining challenge is carrier interference, if the interfering signal is stronger than the MFSK signal. In effect, the carrier "captures" the symbol decoder on every successive symbol, depriving the FEC decoder of useful data. The planned strategy to combat this is the development of an automatic notch filter to follow the FFT, since the FEC decoder provides copy with missing bits.

Narrow filters are not required for MFSK, any more than for PSK31, since the dynamic range of the detector is high and the detector is not sensitive to out-of-band signals. A narrow filter is however useful to reduce desensitization and AGC pumping in the receiver.

The User Software

The IZ8BLY development software used to prove the specification will run some 80 different combinations of modulation (8 to 64 tones), baud rate (4 to 32 baud) and different FEC regimes from none to R=1/6 and K=9! From this has evolved a relatively simple Windows 95™ user program, offering the two MFSK modes, MFSK16 and MFSK8. The software also includes PSK31 and PSK63F, Nino's 62.5 baud full-time FEC PSK mode. Like the MFSK modes, the FEC in PSK63F is

[14] See http://www.qsl.net/zl1bpu/MFSK/simul.html for a comparison of several popular modes.

transmitted on a single data stream, so avoids the need for QPSK. PSK63F is relatively immune to polar flutter and is a great mode for short haul DX, local QSOs and for beginners, since it is simple to tune.

The modes are chosen from a simple menu. The software includes many useful pre-programmed "metacommands" that can be typed in or used in conjunction with user-defined buttons. Some of these metacommands are associated with the built-in log book, so it is possible to fill in the log data while receiving, and at the touch of a button reply with the signal report and relevant station information.

As well as the usual split window setup, the software has an excellent FFT-based waterfall tuning display, with point and click tuning. The tuning resolution is 1 Hz. There are a number of other excellent "meters", including a PSK31 style "Phase scope" which also indicates MFSK frequency error, a small "Bit shape" oscilloscope display, and perhaps the most interesting of all, a "Clock Alignment" display. This display is a small waterfall, with time in both directions – five sequential symbols are displayed horizontally (brightness indicates symbol amplitude), while vertically about ten seconds worth of these events are shown. When conditions are stable, five black and white bars appear. If the receiver and transmitter clocks are different, the bars tend to slope. As the ionosphere alters the timing, the bars slide left and right, or even jump if the path changes suddenly. When the path is very unstable, or the signal is very weak, the bands become very jagged. The display clearly shows what the ionosphere is up to.

This user program, called "Stream" is now available in beta-release form.[15] The latest versions are fully functional and well documented. The software will operate on a Pentium 75 or faster, and requires a Windows™ compatible sound card. The algorithms, codes and alphabet are public domain,[16] making the mode legal to use in most countries of the world. It is intended to release source code, in order to promote development of improved software from other programmers, and for other platforms such as LINUX. There is also an email reflector[17] for users to swap notes, report performance and problems, and to set skeds.

At the time of writing, there are some 200 MFSK16 users world-wide, from countries as diverse as Finland and Fiji. Most DX activity is on 20m and 17m, with local activity on 80m and 40m.

Bibliography

Claude Shannon, *A Mathematical Theory of Communication*, Bell Sys. Tech. Journal, 1948
Robert Fano, A Heuristic Discussion of Probabilistic Decoding, IEEE Trans. Inform. Theory 1963
Andrew Viterbi, Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm, IEEE Trans. Inform. Theory 1967
Peter Martinez G3PLX, *PSK31 Fundamentals*, http://aintel.bi.ehu.es/psk31theory.html
Peter Martinez G3PLX, *PSK31: A New Radio-teletype Mode with a Traditional Philosophy*, Radio Communication 1999
http://det.bi.ehu.es/~jtpjatae/pdf/p31g3plx.pdf
Tom McDermott N5EG, Wireless Digital Communications: Design and Theory", TAPR 1996
Chip Fleming, A Tutorial on Convolutional Coding with Viterbi Decoding, Spectrum Applications 1999.
http://pweb.netcom.com/~chip.f/viterbi/tutorial.html
Phil Karn KA9Q, *TAPR Error Control Coding Seminar*, March 1995,
http://www.tapr.org/dsp/Texas_Instruments/dsp93/stlouis.wrkshp.95/95ECCSeminar.PhilKarn.pdf
Phil Karn KA9Q, *Convolutional Decoders for Amateur Packet Radio*, http://people.qualcomm.com/karn/papers/cnc_coding.html
L. Devaux and F. Smets *A Seven Tone Radio Printer*, Electrical Communication, 1937.
R. Maniet, *Les Téléimprimeurs, Téléchiffreurs et Transcodeurs ACEC - Système Coquelet*,
ACEC-Revue, No 3-4, 1970.
H.K. Robin OBE et al, *Multi-tone signalling system employing quenched resonators for use on noisy radio- teleprinter circuits*, Proc. IEE Vol. 110 No. 9, September 1963.

[15] From the MFSK website http://www.qsl.net/zl1bpu/MFSK

[16] On the MFSK website.

[17] MFSK@egroups.com. Subscribe by sending an email to MFSK-subscribe@egroups.com

A Zero-IF digitized Despreading Scheme Without PN Code Synchronization Recovery

Hong Guo , Feng Guo
Xidian Univ.MCN Group
119#, 2TaiBaiNan Road, Xi'an, Shaanxi, P.R.China (zip code:710071)
hguo@mcn.xidian.edu.cn

Tao Duan
Airforce Engineering Univ.
111#, 1FengHao Road, Xi'an, Shaanxi, P.R.China (zip code:710077)
duantaosohu@sohu.com

Abstract

Zero-IF is a important method to digitize the despreading.In this paper,a fast zero-IF digital despreading scheme without PN code synchronization recovery is given,which is based on performance analysis to chip rate sampling and multiple chip rate sampling in AD convertor.

Keywords: Zero-IF, despreading , chip rate , sampling , synchronization recovery

Introduction

The 3G telecommunication system standard make the broadband CDMA the main tendency.However,we get higher data rate at expence of more complex and expensive transeiver,especially the despreading receiver.

Digitized spreading/despreading is the most significance development.To direct sequence spread spectrum(DSSS) it means AD convertor(ADc),Digital Match Filter(DMF),ASIC and DSP are used.Most digitized despreading schemes are tecnologicially realized based on zero-IF,especially analog zero-IF for its lower complexity.In this scheme AD converting is a process which sample and quantize baseband chip wave stream,and the sampling determines the post circuit and its performance,so accuracy of the sampling is very important.

In digitized despreading schemes two sampling methods are used---chip rate sampling and multiple chip rate sampling.Sampling clock of the former equal to the chip rate,that is to say receiver just sample the chip wave once in a chip period,but the latter sample two or more times.We can use the former method to design a simplest DMF,but with a accurate PN code synchronization recovery unit as an offset.However the latter can provide accurate sampling under any phase difference between local sampling clock and chip stream,this means that we have possibility to abandon the more complex PN code synchronization recovery unit,certainly,scale of the DMF will expand correspondingly.

In this paper,we make analysis to and comparation between the chip rate sampling and the double chip rate sampling method.We find that the latter can provide enough Signal-Noise Rate(SNR) in general utilization.Based on this discovery,we develop a double chip rate sampling digitized despreading scheme without PN code synchronization recovery unit.

1 Zero-IF DSSS receiver

Zero-IF signal can be generated by analogue or digitized method.The latter demand that Adc(s) is(are) applied in IF band,which make system difficult to realize.In main applications the former is adopted.Figure 1 illustrates a typical analogue zero-IF DSSS receiver.

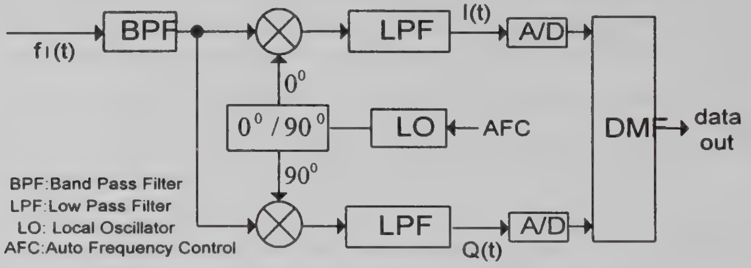


Figure 1 A typical analogue zero-IF DSSS receiver

2 Chip Rate Sampling and Multiple Chip Rate Sampling

Bit Error Rate(BER) of a DSSS system in radio channel can be expressed asit

$$P_e = erfc\left(\frac{1}{\sqrt{\frac{K-1}{3N}+\frac{1}{SNR}}}\right) \qquad 1$$

Where $erfc(x) = \frac{1}{\sqrt{2\pi}}\int_x^{\infty} e^{-y^2/2}\,dy$, K is the number of PN codes that arrived at radio channel at same time and N is the length of PN codes(given that all codes' length are same).To make the explanation more clear,we select K=5,20,35 and N=63 and take them into Equation (1),we can get a set of relations between P_e and SNR illustrated in Figure 2.We chose K=20 as a example and easy to find that when $SNR \approx 5dB$ P_e will reach 10^{-3} OM.

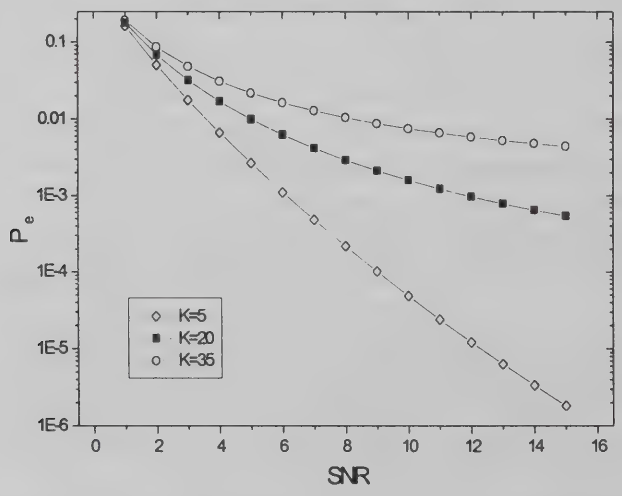


Figure 2 $P_e \cdots SNR$ **curve**

When chip stream is asynchronous to the local PN code clock,a key problem is that how to set the ADc to sample a chip sample as great as SNR.That is so called synchronization.Inserting PN code synchronization recovery unit is a way,but is more hemogeneous.To system illustrated in Figure 1,assuming $d(t)$=1 or 0 is data bits,if modulation is DPSK,then IF input signal is $f_I(t) = N\sin(\omega_T t + \varphi)d(t)$,where N is amplitude(given as 1 without influence) and ω_T φ is received carrier frequency and phase deviation respectively.Assuming output of the LO is $\cos(\omega_L t + \theta)$,where ω_L, θ is frequency and phase respectively and other conditions are perfect,then the zero-IF signal passed the LPF is

$$I(t) = [\sin(\omega_T t + \varphi)\cos(\omega_L t + \theta)d(t)] * h(t)$$

$$= [\frac{1}{2}\sin(\Delta\omega t + \Delta\varphi)d(t)] * h(t) \qquad 2$$

and $$Q(t) = [\sin(\omega_T t + \varphi)\sin(\omega_L t + \theta)d(t)] * h(t)$$

$$= [\frac{1}{2}\cos(\Delta\omega t + \Delta\varphi)d(t)] * h(t) \qquad \_3$$

Where $\Delta\omega = \omega_T - \omega_L, \Delta\varphi = \varphi - \theta$,AFC can drive ω_L trend to ω_T,so $\Delta\omega \approx 0$,thus suppressed the mirror.Assuming $\Delta\varphi$ constant is reasonable, $*$ denote convolution, $h(t)$ is the time impulse response of the LPF(we deem it is ideal, i.e.it has rise cosine spectral property).From Equation (2) and (3) we see that $I(t)$ is a function of rise cosine spectral property expressed as Equation (4),so is $Q(t)$ which has same wave as and $\frac{\pi}{2}$ phase diviation to $I(t)$.

$$I(t) = A\frac{\sin(\pi\omega t)}{\pi\omega t}\frac{\cos(\alpha\pi\omega t)}{1 - 4\alpha^2\omega^2 t^2} \qquad 44$$

Where ω is data rate,α is roll down coefficient and A is amplitude.If $\omega t = T$ $\alpha = 1$,then Equation (4) can be simplified as

$$I = A\frac{\sin(\pi T)}{\pi T}\frac{\cos(\pi T)}{1 - 4T^2} \qquad 55$$

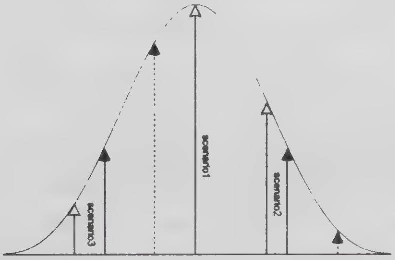


Figure 3 Asynchronous sampling scenarios

We design chip wave can provide maximum SNR which drive P_e to 10^{-4} OM,accroding to Figure 2 the SNR is about 12dB.If chip rate sampling is applied without PN code synchronization recovery unit,just one of three scenarios will appear,which is explained by hollow arrow in Figure 3.

scenario 1: Sampling point just place at the symmetry axis of the chip wave(let this point is

$T=0$).From Equation (5) we can write that

$$I = A\frac{\sin(\pi T)}{\pi T}\frac{\cos(\pi T)}{1-4T^2}\Big|_{T=0} = A \qquad 6$$

In this scenario the DMF can get 12 dB input SNR and output data stream with 10^{-4} OM.

scenario 2: The phase difference between sampling point and the chip wave is less than sc/2,but far higher than zero.The SNR of this sample is less than 12dB,but higher than 5dB,so can make system to get 10^{-3} OM output.

scenario 3: The phase difference between sampling point and the chip wave verge to zero.It is obvious that at this time the receiver can not despread chip stream correctly.

Up to now the key factor can be generalized as where is the bound between the scenario 2 and 3.We notice that I is a monotone function,so this problem equivalent to find the value of T which make I to satisfy $SNR = 5dB$ (let this value of OLE_LINK10105000001000000100000000576f72642e446f63756d656e742e3800T is T_B).We let N_0 denote average noise power,thus the SNR can be defined as

$$SNR = 10\lg\left(I^2/2N_0\right) \qquad \_7$$

Let $SNR_H = 12dB$, $SNR_L = 5dB$,from Equation (5) (7) we can show that

$$\frac{A^2}{2N_0} = 10^{SNR/10} \qquad 8$$

and

$$SNR\Big|_{T=T_B} = SNR_L = 10\lg\frac{A^2\left[\frac{\sin(\pi T_B)\cos(\pi T_B)}{\pi T_B(1-4T_B{}^2)}\right]^2}{2N_0} \qquad 9$$

TakeTa8 into in99n we get

$$\frac{\sin(\pi T_B)}{\pi T_B}\frac{\cos(\pi T_B)}{1-4T_B{}^2} = \sqrt{10^{\frac{SNR_L-SNR_H}{10}}} \qquad 100\_$$

We solve Equation (10) and get the bound: $T_B \approx 0.5358$.From (5) we know when $T \approx 1$,I=0, i.e.if duration of the chip wave is 2 and phase difference between sampling point and the chip wave is uniform distribution between 0 and 2if,the sampling point must fall into the scope that the maximun and minimum T is +0.5358 and –0.5358 respectively to enable P_e to touch 10^{-3} OM.It is clear to see chip rate sampling without PN code synchronization recovery is very easy to sample incorrectly,the probability is about 1-0.5358 = 46.42%.

If we increase sampling rate to double chip rate,as solid arrow showing in Figure 3,the result will be dramaticly changed.It is in evidence that the higher one of the two samples reach to its minimum just when the two sample equal(showed as solid line solid arrow in Figure 3),in any other scenario there is always a sample of the two higher than the worst minimum(showed as dashed line solid arrow in Figure 3).By the derivation method just as above-mentioned we can know that the worst minimum appears when two sample happen to place at T=+0.5 and T=-0.5 respectively.The conclusion we have got tell us in this worst scenario P_e still can sure 10^{-3} OM.That means error sampling probability can be zero.

3 Fast Zero-IF digitized despreading scheme based on double chip rate sampling

Analysis gave above show we that double chip rate sampling can provide satisfactory result under arbitrary asynchronous phase.It hint the PN code synchronization recovery unit can be omitted and we are able to design a fast despreading sysytem.

We select 32-tap DMF as the basic despreading unit.PN code length is 32 bits.Aimed to realize double chip rate sampling,we introduce a 64-tap serial shift register whose length is double the PN code's.We divide the register into two groups without modification of its connection.The two groups together with a common 32-tap reference register form two 32-tap digital correlators.The concrete division method is the registers with odd serial number from input port form group 1,the even ones group 2.The group 1 store the former of the two samples of one chip wave(i.e.the sample shifting into register advanced) and group 2 the latter.It is obvious that in one of utmost 127 samlping clock periods group 1 and 2 just right store the PN code's two groups samples respcetively,and at least one of the two correlators will output correct correlation peak instantly.If two peaks are outputed at same time,the higher will be reserved and used for demodulation.All mentioned above are illustrated in Figure 4.

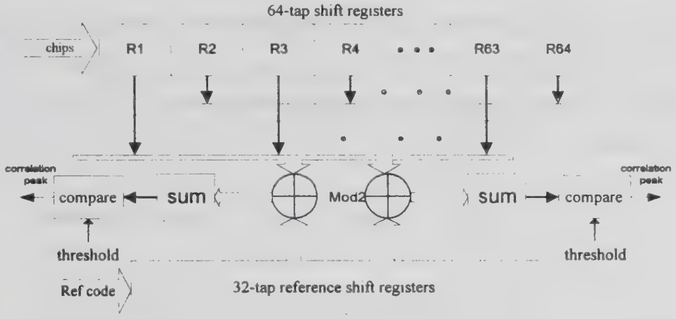


Figure 4 Double chip rate sampling fast zero-IF digitized despreading scheme

We designed and simulated the scheme in a FLEX10K10 EPLD of Altera.The development enviroment is Max+PLUSLULUversion8.3ve.The results of the simulation is showed in Figure 5 and it tell us the scheme is workable.

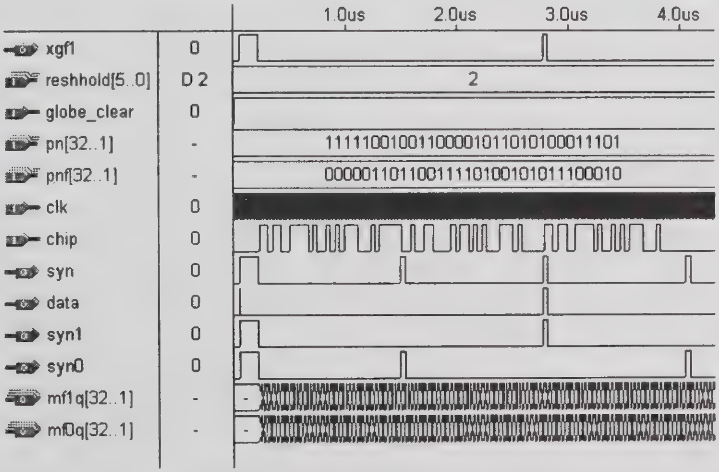


Figure 5 Simulation results

Conclution

We give a zero-IF digitized despreading scheme without PN code synchronization recovery in this paper.Through the theorial analysis is developed by some concrete data,generalization of the result is certain.The key factor is we want to give a analysis concept and method.The scheme passed the chip level simulaqtion,but still needs more examination in real radio channel.

Reference

[1] Jack M.Holtzman.(1992)"A Simple,Accurate Method to Calculate Spread-Spectrum Multiple-Access Error Probabilities" IEEE Trans Commun,vol.40, NO.3,MARCH

[2] Advance product information Stel2000A. Stanford Telecom. 1994

[3] Harris data book Hsp3824 1996

PIC-et Radio II: How to Receive AX.25 UI Frames Using Inexpensive PIC Microcontrollers

by John A. Hansen, W2FS
State University of New York
49 Maple Avenue
Fredonia, NY 14063
hansen@fredonia.edu

Abstract: This paper provides step by step documentation of how to decode AX.25 UI frames using inexpensive PIC microcontrollers. It is designed primarily for those who wish to receive packet radio UI beacons from point to multipoint communications. The article assumes some knowledge of programming concepts and PIC microcontrollers.

Keywords: AX.25, UI Frames, PIC Microcontrollers

Introduction

At the 1998 ARRL/TAPR Digital Communications Conference I presented a paper describing how to encode AX.25 frames using a PIC microcontroller so those frames could be transmitted via Amateur Radio.[1] However, having learned to transmit packet using cheap chips, it was only natural that people would want to know how to receive packet frames as well. The first effort made in this direction that I know of was some assembler code developed by Byon Garrabrant, N6BG. A number of packet receive routines (by Byon and others) are currently available on the TAPR PICSIG software FTP site (ftp.tapr.org/picsig/software). The purpose of this paper is to explain the basics of how PIC-based packet receive firmware works.

Note that this paper will not provide you with a working packet receive system. If that is what you are seeking, simply use the code provided on the TAPR FTP site. Instead, my purpose is to describe the theory behind how PIC based packet receive routines work so that you will be able to create your own code to decode AX.25 frames. In keeping with this goal, I've tried to make the code that is presented here as simple as possible. To do this, I made a number of simplifications. First, all code examples here are written in C (my code is designed for the CCS C compiler, because it is relatively cheap).[2] I did this because I think that regardless of whether you are planning to develop your project in C or assembler, C is somewhat easier for most people to follow. Secondly, the code presented here is designed to receive packets that are no longer than 40 bytes (including the header). I used this approach because I wanted to use only a PIC16F84 microcontroller and an MX-614 modem chip. Significantly longer packets would require some external storage, but the 16F84 has adequate on-board storage to handle relatively short packets. A number of packet receive implementations have been realized that use external storage and can handle longer packets. For example, Mike Berg N0QBH has used a Ramtron FRAM chip as storage for longer packets. I have used a 32K static RAM chip in my PIC KISS TNC to provide both transmit and receive storage.[3] However, limiting the receive data to 40 bytes means that

[1] Hansen, John A., "PIC-et Radio: How to Send AX.25 UI Frames Using Inexpensive PIC Microprocessors" in *17th ARRL and TAPR Digital Communications Conference* (Newington, CT: ARRL, 1998) p. 29. Also available as DCC.ZIP at ftp.tapr.org/pub/wa0ptv.

[2] www.ccsinfo.com

[3] Hansen, John A., "An Inexpensive KISS mode TNC" forthcoming in *QST*.

we need not include here routines for writing and reading to storage. Finally, the code presented here is designed to receive UI frames. The results presented here could be extended to cover the range of AX.25 frames, since the principles involved in receiving the data are the same in any case. To make this code relatively simple, however, this paper will present code that merely receives the frame, assumes it is a UI frame and formats it and pumps it out a serial port to a terminal. The construction of UI frames is not covered here in detail. For a discussion of how to build up UI frames, see my 1998 DCC paper.

Receiving Bits

The essence of receiving packet is receiving data bits. While data is transmitted over the air as a series of tones (1200 Hz and 2200 Hz), a bit of information is not represented by the frequency of the tone, but rather it is represented by whether there is a change in tone or not. A shift in tone (either from 1200 to 2200 Hz or vice versa) represents a zero, while no shift in tone represents a one. This project assumes that modem functions are done by another chip (in this case the MX-614) so that an input pin on the PIC sees a "high" (+5 volts) when there is 1200 Hz tone and a "low" (0 volts) when there is a 2200 Hz tone. We will begin the discussion of receiving packet frames with the code that is used to receive a bit. I use the function bitin() to accomplish this:

```
int bitin(){                               //function to read a bit

static int oldstate;                       //oldstate retained between runs of this function
int k;

     for (k=0;k<121;k++){                  //this loop allows 838 us to go by.  If no state change, bit is 1
      if (input(rcvPin) != oldstate){      //if state has changed
           oldstate = input(rcvPin);       //update oldstate
           delay_us(430);                  // move to halfway thru the next bit
           return 0;                       //return 0 if state changed
        }//end of if
     }//end of for
     return 1;                             //return 1 if state did not change
}//end of bitin()
```

The PIC pin that is connected to modem chip is referred to as "recPin". The main purpose of this routine is to look for state changes on the input pin. The function "input(recPin)" is built into the CCS compiler and returns a 1 when the pin named rcvPin is high and a 0 when the pin is low. Two variables are used here, k (to count the number of times through the loop) and oldstate, which keeps track of the previous state of recPin. Oldstate is a *static* variable so that it won't lose its value between times when this function is called. It cycles through the 'if' loop 121 times to see if the state has changed from it's old value. Assuming you are running the PIC at 10 MHz, 121 times through this loop will take 838 microseconds (μs) to complete.[4] In a 1200 baud transmission, each bit lasts 833 μs (1,000,000/1200). Thus if 838 μs go by with no change of the state of recPin, we must be looking at a 1 rather than a zero. So the function returns a 1.

[4] The simulator function of the PIC development environment MPLAB allows you to precisely time events while the code is running. MPLAB is available free of charge from www.microchip.com.

If, on the other hand, a state change is detected during this time period, the function returns a zero. If it finds a zero, the function will also introduce a delay of 430 μs. This is an important point. Packet is an asynchronous communications system. While a bit lasts 833 μs, there is no way to know where a particular bit would start and stop other than by looking for those instances when the tone changes. At the moment the tone changes, we know where we are in the bitstream. Thus we recalibrate the receiver on every tone change. Adding 430 μs right after a tone change throws us just past the middle of the next bit. Consequently every time a zero (tone change) is detected the receiver is reliably placed at a spot that is just after the middle of the following bit. If this were not done, small errors in timing would add up as we received more and more bits and eventually we might miss a bit altogether. Packet radio transmissions are designed to ensure that a zero will be transmitted at least once every six bits. Thus the receiving routine is recalibrated at least every five milliseconds.

The function bitin() receives one bit of the packet bitstream. The trick is to make sure that the function is called sufficiently often that no bits are missed. In addition, we must know how to process the accumulated bytes when an entire packet is received.

Let's Play Capture the Flag

You cannot decode a packet frame unless you receive the entire frame. This is because the frame contains a two byte frame check sequence (FCS) that must match the value transmitted by the sending station in order for the packet to be valid. Hence it does no good to begin receiving in the middle of a frame. When the program first starts, therefore, it must begin by looking for the flags that indicate the beginning of a frame. These flags are simply the byte 01111110 in binary (7E in hexadecimal notation). Most (though not all) TNCs send this series of bytes during the entire transmit delay (TXDelay) period, but all must send at least one flag to indicate the beginning of the frame. The problem is that when you turn your packet receive system on, you don't know for sure that you are receiving the beginning of a packet. You don't even know for sure that you are receiving the beginning of a byte. So you must continually receive and check bits until you find the 01111110 sequence. Because of bit stuffing (discussed below) the only time you will receive six ones in a row is when you are receiving a flag.

Here is the code to look for the first flag:

```
int cbyte = 0;                          //initialize
while (cbyte != 0x7e){                  //find the first flag
        shift_right(&cbyte,1,bitin());  //add a new bit to the left of cbyte, discard right bit
 } //end of while
output_high(LED); //turn on the DCD light
```

The line that starts with shift_right may seem a little strange because it is a function provided by the CCS compiler, not by the C language. What it is says is as follows. Get a new bit from the receiver (using the function bitin()). Move each bit in the byte called cbyte one position to the right. Then take the new bit and append it to the left end of cbyte. So, for example, if cbyte was 10001000 before this line of code was run and if the new bit obtained by bitin() was a 1, the resulting cbyte would be 11000100. The bit on the extreme right gets bumped off the end and lost. The purpose of this is to retain a memory of the previous seven bits that were received and to keep getting new bits until the most recent 8 bits match the pattern 01111110 (7E). When this occurs we know that we have found a flag. At that point we turn on the DCD light using the CCS built in function output_high(LED), to indicate that data is being received.

Well, almost. It seems the MX-614 chip sends a random series of ones and zeros whenever it is not actually receiving audio tones. As a result, sometimes it just happens to send the 7E byte. When this happens the DCD light will come on even though no data is being received. However, whatever random series of bits that is received afterwards will almost certainly not pass the FCS check, so these "errors" will only show up as a flickering of the DCD light, not as the reception of bad data.[5]

After the first flag has been received, the next byte may be a flag, or it may be the beginning of the actual data. However, from here on out, we can grab the data eight bits at a time because we know that we are at the beginning of a byte. So instead of examining the byte after every bit is added, we can simply repeat the procedure of grabbing a bit and shifting it into cbyte eight times:

```
int i;
int bte[40];
While (cbyte = = 0x7e){                       //find the other flags
        for(i=0;i<8;i++){                     //repeat this 8 times
                shift_right(&cbyte,1,bitin()); //add a new bit to the left of cbyte, discard right bit
        }                                     //end of for
}                                             //end of while  -- now at end of all the flags
bte[0] = cbyte;                               //you've now got the first address byte
```

The while loop continues to execute until it finds something other than a flag. When the loop exits, the value in cbyte will be the first value of address portion of the AX.25 frame. I used a forty element array called bte to hold the actual data from the packet. Thus the value in cbyte is assigned to the first (zeroth) element of the bte array.

Collecting the Data

From here we need to continue to collect data until another flag is encountered telling us that we have reached the end of the packet. However, there is one more complication. As noted above, the AX.25 protocol ensures that there will be a zero (and hence a tone change) at least every six bits. Furthermore, a flag is the only instance in which six ones in a row are allowed to occur in the data stream. If six ones were to occur in the data itself, the protocol requires that a 'zero' bit be inserted after the fifth 'one' bit. This procedure is referred to as "bit-stuffing". For example, suppose the data included the two byte sequence 00001111 11110000 (in hexadecimal notation this would be 0F F0). In that case, a zero would be added after the fifth one in the sequence, so now seventeen bits would be

[5] Note that in a "real" receiving system you might want to add a mechanism for providing a limit on the number of bytes of random data that the unit would interpret as real data before testing (and presumably failing) the FCS check.

transmitted as follows: 00001111 101110000. This extra zero needs to be removed on the receive end in order for the data stream to be the intended sequence of 0F F0. So while receiving it is important to monitor the bit stream for any zero that occurs after five consecutive ones. Of course, if the next bit after five consecutive ones is another one, we know that we have found the flag that ends the packet and it is time to stop receiving data.

Note that it is not necessary to monitor the first byte for bit stuffing. The reason is that each byte of the address field (except the last byte) must end in a zero. Since the address field must be at least fourteen bytes long, you really don't need to worry about bit-stuffing for the first few bytes.

Here is the code that is used to collect the rest of the data:

```
int test, newbit, numbyte, ones;
test =0;
numbyte = 1;                                  //we already collected 1 byte
while (test != 1){                            //do this until the flag at the end of the packet
        for(i=0;i<8;i++){                     //collect 8 bits
                newbit = bitin();             //get a bit
                if (newbit = = 1) (ones++);   //increment the ones counter
                        else (ones = 0);      //if bit is a zero, reset the ones counter
                if (ones = = 5) {             //removes bit stuffing
                      test = bitin();         //get the next bit but don't add it to cbyte
                      ones = 0;               //reset the ones counter
                }
                shift_right(&cbyte,1,newbit); //append the new bit to cbyte
        }                                     //end of for
        if (test = = 0){
                bte[numbyte] = cbyte;         //add cbyte to the array
                numbyte++;                    //increment the number of received bytes
        }
}//end of while
```

Test is a variable that is a one if a flag is encountered and a zero otherwise (it could be boolean rather than int). This snippet of code collects each bit and tests to see if it is a zero or a one, and appends it to the value of cbyte. The variable 'ones' keeps track of the number of consecutive ones that have been received. If the new bit is a one, it increments the ones counter; if it is a zero it resets the ones counter to zero. If five consecutive ones are ever detected, an additional bit is retrieved. This additional bit is not added to the current byte, but rather it is put in a variable called 'test'. If value of test is a one, it means that we have received a flag and the data transmission is done. If test is a zero, it means that bit stuffing occurred and the value is discarded. As each byte is completed, it is added to the array bte and the number of bytes (stored in numbyte) is incremented.

The while loop continues until a flag is encountered. When the loop exits, all of the data in the packet is stored in the array bte and the number of data bytes that were received is stored in numbyte.

Checking the FCS and Formatting the Output

All that remains is to check to see whether the received data is valid and to format it and route it out the serial port if it is valid. The following line of code performs this function:

```
if (fcscheck()) (printout());                          //if the fcs checks output the packet.
```

This code calls two functions, fcschcheck() to determine if the data is valid, and, if it is, printout() to route it to a serial port. The AX.25 protocol calls for the use of a cyclical redundancy check that it refers to as a frame check sequence (FCS). Two bytes are appended to the end of each frame after the data that are the FCS values. At the receiving end, to ensure the integrity of the data, we must calculate the FCS and compare the values with those that are transmitted in the packet itself (and have been calculated at the transmitting end). The fcscheck() function that performs this check is virtually identical to that included in my 1998 paper on transmitting AX.25:[6]

```
short fcscheck(){                                          //computes the fcs
int i,k,bt,inbyte, fcslo, fcshi;

  fcslo=fcshi=0xFF;                                        //intialize FCS values
  for (i = 0;i<(numbyte-2);i++){                           //calculate the FCS for all except the last two bytes
      inbyte = bte[i];
      for(k=0;k<8;k++){                                    //perform this procedure for each bit in the byte
            bt = inbyte & 0x01;
            #asm                                           //embedded assembly language route to do a 16 bit rotate
                BCF    03,0
                RRF    fcshi,F
                RRF    fcslo,F
            #endasm
            if (((status & 0x01)^(bt)) ==0x01){
                fcshi = fcshi^0x84;
                fcslo = fcslo^0x08;
            }                                              //end of if
            rotate_right(&inbyte,1);                       //set up to do the next bit
      }                                                    //end of for
  }
  fcslo = fcslo^0xff;
  fcshi = fcshi^0xff;
  if ((bte[numbyte-1] == fcshi) && (bte[numbyte-2] == fcslo)) {
      return 1;
  }
     else return 0;                                        //if the computed values equal the last two data bytes
}                                                          //return a 1, otherwise return a 0
```

[6] For a discussion of how this seemingly arcane calculation is performed, see Morse, Greg "Calculating CRCs by Bits and Bytes" (*Byte,* Sept 1986, pp. 115-124).

All that remains is to format the output and route it out the serial port. Doing this requires an understanding of the construction of AX.25 UI frames, but otherwise is fairly trivial. Here is the printout() procedure that I used to accomplish this:

```
void printout(){                                        //function to display the received packet
int i,L,m,temp;

    for (m=7;m<13;m++){                                 //print the source callsign
        if (bte[m] != 0x40) (putc(bte[m]>>1));          //note spaces (40) are not printed
    }
    putc('-');
    putc(((bte[13] & 0x7F)>>1));                        //print source SSID
    putc('>');
    for (m=0;m<6;m++){                                  //print the destination callsign
        if (bte[m] != 0x40) (putc(bte[m]>>1));
    }                                                   //end of for
    putc('-');
    putc(((bte[6] & 0x7F)>>1));                         //print the dest SSID
    L = 7;

    if ((bte[13] & 0x01) != 1){                         //print any path that may exist
        do{
            putc(',');
            L=L+7;
            for (m=L;m<(L+6);m++){
                    if (bte[m] != 0x40) (putc(bte[m]>>1));
            }                                           //end of for
            putc('-');
            putc(((bte[(L+6)] & 0x7F)>>1));
        }while ((bte[L+6] & 0x01) != 1);
    }                                                   //end of if
    putc(':');
    putc(' ');
    L=L+9;                                              //add 9 to move past the last callsign, cntl and PID
    while (L< numbyte-2){                               //print the text (not including fcs bytes)
        putc(bte[L]);
        L++;
    }                                                   //end of while
    printf("\n");                                       //add carriage return/line feed at the end
}
```

The variable L is used to walk through the data. The AX.25 protocol calls for the destination callsign to be included first followed by the source callsign. Note in the printout these are placed in the opposite order, since this is the de facto standard on the monitor function of most TNCs. Thus a packet with the text “Test” addressed to APRS from W2FS-2 via RELAY will show up here as:

W2FS-2 > APRS, RELAY: Test

The AX.25 protocol requires that the callsigns in the data contain exactly seven bytes (padded with spaces if necessary) and be left shifted by one bit. The code above undoes these changes. The SSID values (such as bte[13]) must be anded with 7F to eliminate the “command/response” bit that is also carried on this byte. There are also two "reserved" bits on this byte that are supposed to be set to one. Conveniently (and perhaps by design) they result in the addition 30 (in hexidecimal) to the SSID value. That is, an SSID of 1 is appears as hex 31 after it is anded with 7F. This is convenient because the hex value 31 is produces an ASCII value of "1" when it is printed out.[7] The control, PID and FCS bytes are not printed.

Conclusion

To put all the pieces together, it is necessary to set up an infinite loop that causes the PIC continuously loop through these functions. In pseudocode it would look something like this:

```
Initialize I/O pins and set modem to receive
While (TRUE){
        Look for the first flag
        Check for additional flags until you have the first byte of data
        Continue receiving data until you reach the terminating flag
        Check the FCS
        Route output to the serial port if the FCS checks
}
```

It seemed almost magical when I first burned this code into a 16F84 and watched the output flow across my screen. What once seemed to require so much hardware can now be accomplished with less than $12 worth of integrated circuits.

[7] The same trick works for all the digits 0 through 9. If one wanted to worry about decoding SSID's greater than 9, some additional coding would be required.

Winlink 2000 ...A Global Ham Message Transfer and Delivery Network

Rick Muething, KN6KB <rmuething@cfl.rr.com>
Vic Poor, W5SMM <vpoor@spacey.net>
Hans Kessler, N8PGR <N8PGR@winlink.org>
Steve Waterman, K4CJX <K4CJX@home.com>

Abstract:

Early digital modes broke from the traditional "live" interactive QSO and fathered the now familiar store and forward message systems we now take for granted in our BBSs and E-mail. The increased demand for mobility in Ham radio has evolved into the need for a reliable, rapid, global message transfer and delivery network that spans both the Ham radio and Internet domains. This paper describes the implementation of the next generation Ham message and information network that allows mobile users global connectivity to other amateurs, Internet mail users and WEB based information while remaining free of the constraints of the "home" BBS. Winlink 2000 is now a fully operational global network serving thousands of mobile ham users and providing reliable connectivity for both text, and binary data.

Key Words:

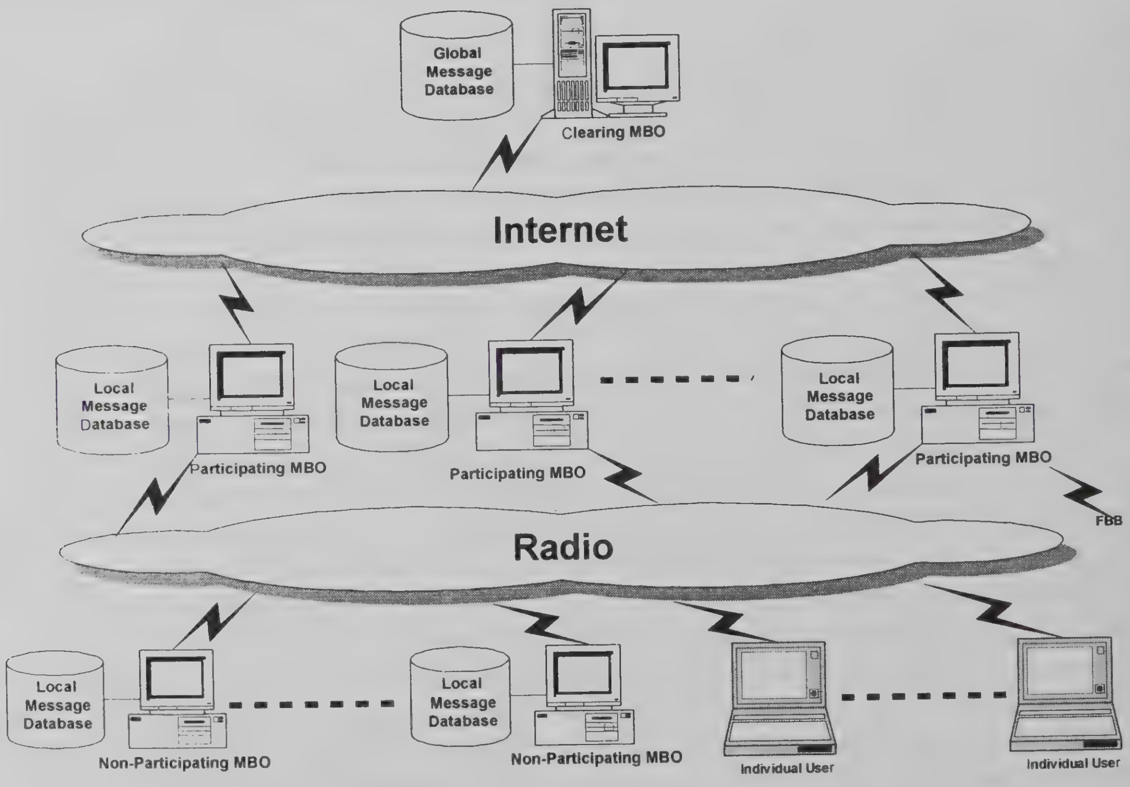
Winlink, Winlink 2000, WL2K, Global Messaging, Internet Gateway, Shared Database, Message Mirroring, Binary Attachments, Wireless Connectivity, Packet, Pactor, Network.

Background and Evolution of Winlink

To fully appreciate a concept like Winlink 2000 requires an understanding of the evolution of its recent predecessors APLink, Winlink, and Netlink. In 1983 Vic Poor, W5SMM, developed APLink, a Microsoft DOS based message store and forward program which allowed HF radio BBS exchange using Amtor. APLink allowed mobile hams outside the range of VHF and UHF Packet stations to connect and route messages similar to the growing popular VHF BBSs of the time. The protocol of these early systems was not error-free and required special consideration to make it workable in the demanding BBS application. The authors of these various systems developed the first communications protocol and coordinated their activities through the "digital committee". One of those authors is Jean Paul Roubelat F6FBB who developed the error-free FBB protocol for use in Packet radio. In 1992 Vic re-wrote APLink to operate under the Windows operating system. The new system was called WinLink. In 1994 ownership of WinLink was transferred to Hans Kessler N8PGR who over time gradually refined and enhanced WinLink to the level we know it today. Winlink, however, still limited the mobile ham to text message traffic with other Ham BBS users. In 1996, after a two-year search, Steve Waterman, K4CJX, an active WinLink SYSOP, found Jim Jennings, W5EUT who agreed to develop an interface into the Internet E-mail system. "NetLink" became the mechanism to add Internet connectivity to Winlink by providing an automated Internet <> Winlink gateway. This NetLink gateway and its NEXUS message format provided mobile hams access to the rapidly expanding Internet E-mail. Jim and Hans added further enhancements to Winlink/Netlink that allowed it to automatically generate weather bulletins based on text based WEB data. Mobile users especially cruisers on the high seas found this not only convenient but a significant safety factor during long passages. Perhaps the biggest shortcoming of this system was the centralization of all functions and message data to a specific Ham BBS (home BBS). If that BBS encountered a problem due to power, equipment, propagation or Internet failure users of that "home BBS" were left without mail service and access to important WX data. The joy and fun of running a BBS could too easily become a rather demanding job of providing reliable full time service on a volunteer basis. It was from this and the expanding need for increased E-mail, weather, and information services that Winlink 2000 (WL2K) was born.

Introduction to Winlink 2000:
In conventional BBS systems (including Winlink and FBB systems) messages are stored in files or a database that is local to each station. Messages in these systems can remain local to the specific BBS or forwarded to other BBSs and (in some systems) Internet users. Users must declare a "home" BBS to establish an address and route by which reply messages can be returned. Winlink 2000 eliminates this "home BBS" concept for any user station that designates itself as "mobile". To do this requires replicating and synchronizing the message database to all BBSs within the WL2K network. When this is done a mobile station may connect to, pick up, and send messages freely from any station in the network. Reply messages (either from radio or Internet users) are able to always use a consistent unique system wide address no matter where the reply is to be picked up. Such a system requires frequent and reliable exchange between all BBSs within the system to insure the database is kept synchronized and fresh. To augment the traditional radio forwarding of message WL2K uses the Internet TCP/IP link as the prime channel for BBS to BBS connectivity. Radio forwarding can be used as a backup if there is a failure in the generally reliable and high bandwidth Internet channel.

Fig 1



WinLink 2000 - Top Level View

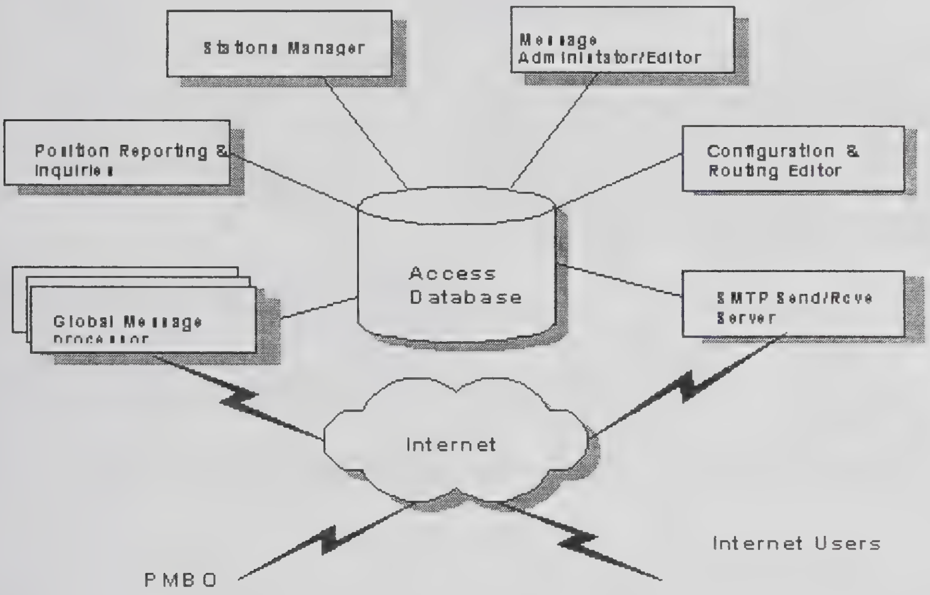
Winlink 2000 Architecture

The Winlink 2000 system architecture is shown in Fig 1. There are four types of nodes in the system interconnected by Radio or Internet TCP/IP channels: CMBO, PMBO, NPMBO, and USER.

CMBO

The CMBO is the Clearing Mail Box Operator: This BBS provides the primary link to the Internet for E-mail services and connectivity to the other PMBOs. The CMBO does not connect directly to end users (it does not support radio channels) and executes the specific WL2K functions outlined in Fig 2.

Fig 2

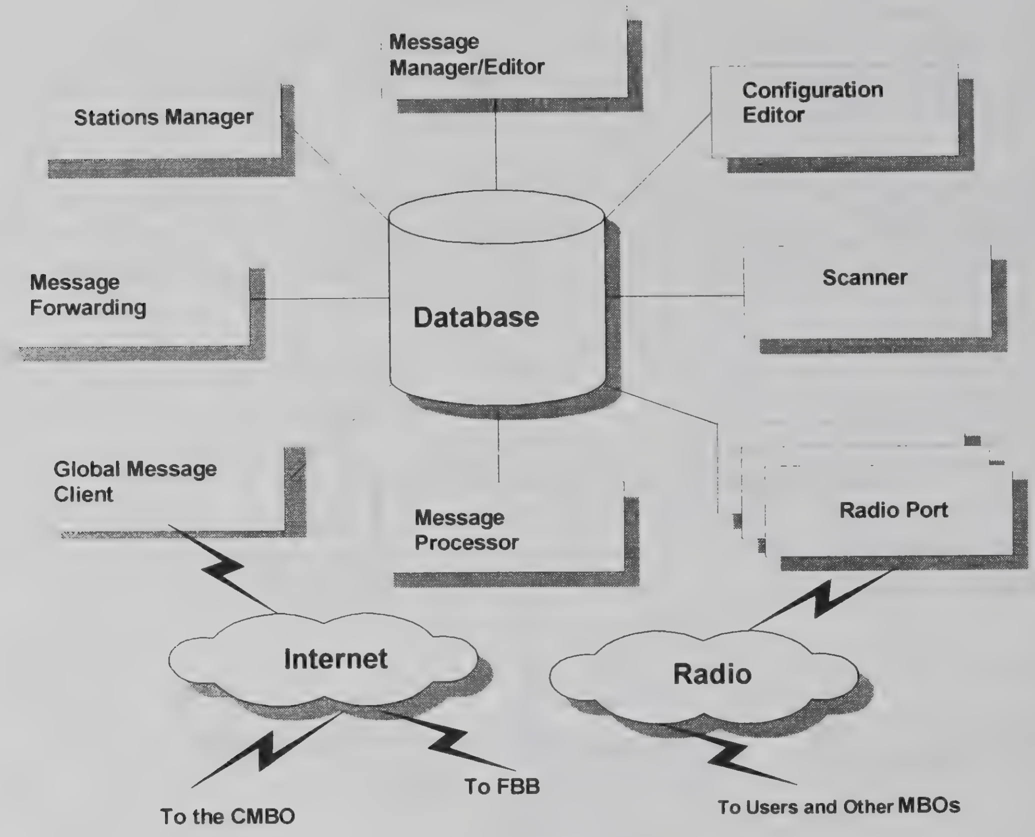


WinLink 2000 - Clearing MBO Tasks

PMBO

The PMBO or Participating Mail Box Operator runs the PMBO specific Winlink 2000 code modules and keeps a local copy of the shared WL2K database (Fig 3). The PMBO provides wireless connection services to the end user and also connectivity (via radio or Telnet) to other Non Participating MBOs (NPMBOs). The wireless links currently include multiple Pactor and Packet channels but are not limited to these modes. Telnet ports provide a convenient and reliable method to move traffic to other systems like FBB that support the Telnet protocol via TCP/IP links. Each PMBO also runs the Inquiry Server that provides on demand weather, local bulletins and other WEB based information in reply to end user Inquiries. Messages picked up by users at one PMBO are signaled by control messages via the CMBO to other PMBOs to clear those messages from the database keeping the PMBO databases in close synchronization. All functions including database repair and backup are automated to allow the PMBO and CMBO to run in unattended mode. Most PMBOs scan multiple frequencies in Pactor I and Pactor II as well as VHF/UHF Packet.

Fig 3



WinLink 2000 - Participating MBO Tasks

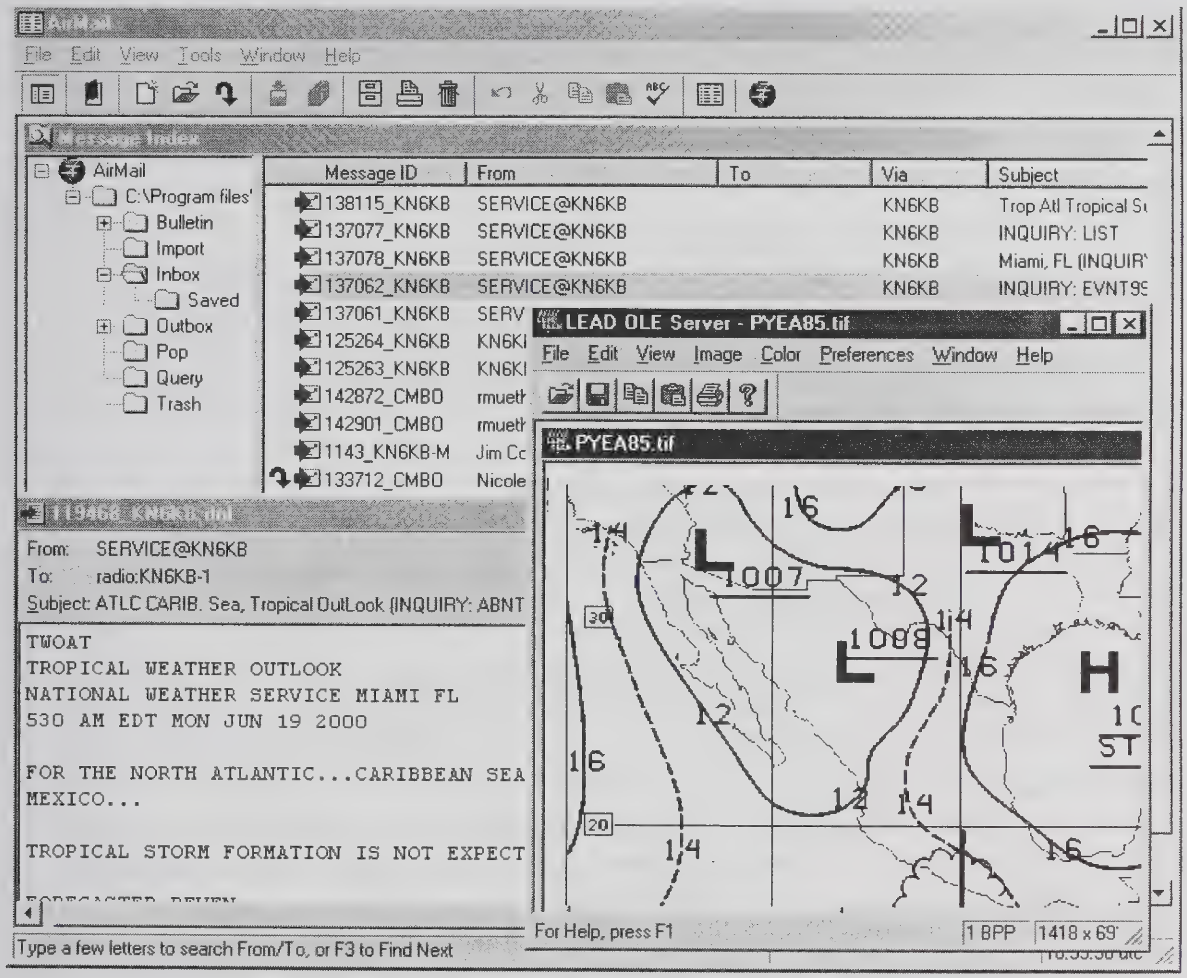
NPMBO

NPMBO nodes connect to the WL2K system via conventional radio or Telnet forwarding using the standard FBB protocol. WL2K provides token routing tables that allow directing any radio or Internet messages to specific NPMBOs based on regional location and the NPMBOs message routing capability. NPMBOs can also forward plain-text E-mail messages to and from the Internet by using the NEXUS message format through any PMBO.

USER

Users (mobile or fixed) connect to the WL2K system either directly via a PMBO or indirectly via a message forwarding NPMBO. The message mirroring (database replication and synchronization) of the PMBO assures that mobile users may pickup and send mail from any PMBO. A WL2K specific client (user interface program) is required to connect to the PMBO for full WL2K functionality. This expanded functionality (summarized in Table 1) includes multiple To and Cc addresses, mixed radio and E-mail addresses, binary attachments and position reporting. Limited WL2K functionality (text messages including NEXUS formatted Internet messages) is available to users through existing FBB compatible clients (traditional BBS systems) via either PMBOs or NPMBOs. Jim Corenman, KE6RK, has expanded his very popular client program AirMail to include full functionality with WL2K. This easy-to-learn user program interfaces with most popular TNCs and provides all the functionality of standard E-mail clients including address and message management in addition to radio specific functions such as automated radio control. AirMail also provides a NMEA GPS interface to allow automated position reporting through WL2K allowing radio and Internet users to easily get position reports from mobile hams. Figure 4 is a screen capture of AirMail's message index along with text and graphical weather data downloaded on demand from WL2K's Inquiry Server.

Fig 4



Message ID	From	To	Via	Subject
138115_KN6KB	SERVICE@KN6KB		KN6KB	Trop Atl Tropical S
137077_KN6KB	SERVICE@KN6KB		KN6KB	INQUIRY: LIST
137078_KN6KB	SERVICE@KN6KB		KN6KB	Miami, FL (INQUIR
137062_KN6KB	SERVICE@KN6KB		KN6KB	INQUIRY: EVNT9
137061_KN6KB	SERV			
125264_KN6KB	KN6K			
125263_KN6KB	KN6K			
142872_CMBO	rmuet			
142901_CMBO	rmuet			
1143_KN6KB-M	Jim Cc			
133712_CMBO	Nicole			

From: SERVICE@KN6KB
To: radio:KN6KB-1
Subject: ATLC CARIB. Sea, Tropical OutLook (INQUIRY: ABNT

```
TWOAT
TROPICAL WEATHER OUTLOOK
NATIONAL WEATHER SERVICE MIAMI FL
530 AM EDT MON JUN 19 2000

FOR THE NORTH ATLANTIC...CARIBBEAN SEA
MEXICO...

TROPICAL STORM FORMATION IS NOT EXPECT
```

Type a few letters to search From/To, or F3 to Find Next

WL2K Implementation

During the early planning stages of Winlink 2000 the system architects anticipated the need to build a flexible scalable system that would provide the high reliability and performance needed in a BBS by capitalizing on the newer 32 bit operating systems and database management tools. The ability to use standard database engines and query languages (SQL) was a significant aid in developing a large and robust shared database system. The current WL2K system is written in Visual Basic 6 for use under Windows 98 and Windows 2000. For cost consideration it uses the Microsoft Jet database engine though it can be readily scaled to a true SQL server for larger implementations if needed. There are a total of 20 separate modules used by the PMBO and CMBO about half of which are utility and database management tools. Multiple instances of the communication drivers are supported which allow a single PMBO to operate multiple radio and telnet channels concurrently. Third party Active X controls were selected to aid in implementing specific Internet Servers and Clients as well as image compression of WEB graphics and support functions such as spell checkers. Standard commercial products are also used in each PMBO to protect against accidental or malicious virus files. The total WL2K effort was done on a purely volunteer basis by the authors over a period of approximately 2 years. Initial on-the-air testing began in October 1999 and by August of 2000 a total of 17 PMBOs and 93 NPMBOs world wide were interfacing to over 1600 Hams and handling in excess of 2000 messages per day with typical message turnaround times on the order of minutes.

Summary of WL2K Features

Table 1 summarizes the major enhancements and additional features of WL2K compared to Winlink and conventional BBS systems.

- **Mobile users (no home BBS required)**
- **Position reporting and tracking (Automatic with AirMail)**
- **Position inquiry from Radio or Internet (current and track)**
- **True E-mail functionality with attachments and forwarding**
  Mixed radio and E-mail addresses with multiple recipients
  User set attachment size limits
  Automatic virus screening
  Junk E-mail protection (lockout)
- **Expanded, interactive Bulletin handling (WX, Help, Info etc)**
- **Forwarding of amateur messages to Internet accounts for inactive or inoperative radio users**

Table 1 WL2K Feature Summary

Future Enhancements for WL2K

Because the system is based on a productive development language like VB and uses standard database management mechanisms expansion to other channels and services is easily embraced. This could include expanded wireless HF, VHF, and UHF protocols as well as amateur satellite connectivity. Due to the intimate connectivity of WL2K with the Internet we anticipate allowing users to access the shared database through a secure Internet connection (WEB browser) as an adjunct to the standard radio channels. Finally, access to reliable and immediate weather and other WEB text, HTML, and image data combined with the robustness of a distributed system make WL2K well suited to support future emergency and disaster communication needs.

For More Information

For additional information on the WL2K system you can visit the WEB sites listed below or E-mail to the authors.

http://winlink.org/k4cjx Steve Waterman's home page containing the latest status of the WL2K Network

www.airmail2000.com Jim Corenman's Web site describing AirMail and related programs. The latest version of AirMail can be downloaded from the site.

Part I: Frequency Offset Acquisition and Tracking Algorithms

Mohamed K. Nezami
Mnemonics Inc.
3900 Dow Rd, Melbourne, FL 32934

I. Introduction

Wireless receivers process signals that bear information as well as disturbances caused by the transmitter/receiver circuits and channel impairments such as fading, interference, and additive white gaussian noise (AWGN). Usually, the receiver knows only some statistical properties of the signal and disturbances. From these statistical properties and using a finite observation of the received signal, the receiver is able to estimate the transmitted data symbols. The receiver makes the decision on the received data using locally generated symbol clock and carrier oscillator, both of which are not referenced to the actual versions used to generate the data at the transmitter. The receiver has to estimate the offset between locally generated carrier and symbol clock to those used at the transmitter. Clock mismatches are labeled as time jitters, while local carrier mismatches are labeled either as phase rotation errors or frequency offset errors. Carrier phase recovery is the process known as carrier tracking, where the receiver estimates the offset between the local oscillator phase and the actual phase of the transmitted carrier. Carrier frequency offset recovery is known as carrier acquisition, and it is the process of estimating the offset between the frequency drift/change of the local oscillator and the actual (received) carrier frequency transmitted at the transmitter. The methods in which the receiver estimates carrier mismatches can be feed forward (open Loops) or feedback (closed loops) synchronization. Feed forward method is characterized by the absence of hang up and cycles slips, and fast and accurate acquisition. Such advantages are absent with conventional feedback synchronizers such as Costas loop. Another advantage of feed forward algorithms is that the synchronizer can be fully implemented using digital signal processing techniques as shown in figure 1.

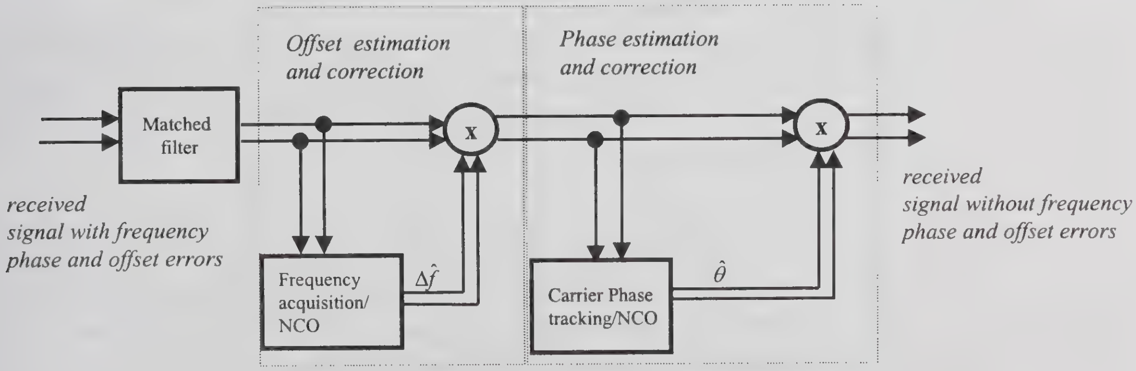

Offset estimation and correction
Phase estimation and correction
Matched filter
x
x
received signal with frequency phase and offset errors
Frequency acquisition/ NCO
$\Delta\hat{f}$
Carrier Phase tracking/NCO
$\hat{\theta}$
received signal without frequency phase and offset errors

Figure 1: Complete Feed Forward Baseband synchronization system

This paper reports an overview of existing as well as presenting new carrier acquisition and tracking systems that are used in modern satellite and personal communication receivers. In particular algorithms that acquire and track the carrier frequency in the absence of symbol timing or a data preamble sequence. The objective is to investigate these algorithms. Then verify their applicability and performance in realistic situations such as wireless hand held cellular and satellite TDMA receivers. The performance evaluation of the algorithms is conducted through simulations and theoretical analysis. The work is performed in two parts. In the first part; we report the mathematical development of these algorithms and their statistical properties and "pull in" range in AWGN channels. In the second part, we present the performance analysis of the recovery process through computer simulation for a variety of applications concerning satellite communications receivers, in particular, techniques proposed that acquires carrier offsets when the initial error (offset) is too large to employ conventional loops (open loops or closed loops). Furthermore, we present a practical means of implementing these algorithms using commercial digital signal processor chips.

II. Carrier Frequency Acquisition and Tracking

In addition to removing carrier frequency offsets, phase errors between the received signal and the local oscillator has to be removed as well. Phase error can result in rotation of the received symbols resulting in degradation of the bit error rate and ultimately degradation in the receiver sensitivity. The BER deterioration as a result of phase mismatches is defined as being the amount of increase in signal power (in dB) that is needed in order to get the same BER performance from a receiver that uses an ideal carrier synchronizer [1,2,3,4,5,6,7,8,9], and given by (1)

$$D = 4.3(1 + E_s/N_0)\sigma_\theta^2 \quad [\text{dB}] \qquad (1)$$

where E_s/N_0 is the symbol energy to noise ratio, σ_θ^2 is the phase variance of the estimated phase out of the synchronizer. For second order Costas loop, the phase variance is given by (2)

$$\sigma_\theta^2 = \frac{B_L T}{E_s/N_0} \quad [\text{rad}^2] \qquad (2)$$

where B_L is the one sided loop bandwidth, and T is the symbol rate (loop iteration rate). Acquiring and tracking carriers using Costas loops for short-burst satellite TDMA signals can prove to be a difficult task. The loop acquisition time as a function of loop bandwidth and the frequency offset to be acquired [6] is given by (3)

$$t_{acq} = \frac{\pi^2 (2\pi\Delta f)^2}{16\zeta(2\pi B_L)^3} + \frac{2\pi}{2\pi B_L} \quad [\text{sec}] \qquad (3)$$

Where ζ is the second order type loop damping factor and Δf is the frequency offset. To achieve faster acquisition, the loop bandwidth in equation 3 has to be increased, while doing so, the phase error variance increases in equation 2, leading to reduction of BER in equation 1. The term on the right-hand side of equation 3 is the tracking time [6], while the term on the left-hand side is the "pull-in" or acquisition time. To demonstrate the loop bandwidth, acquisition timing and BER degradation, let us assume a frequency offset due to Doppler as 1 kHz and a ζ = 0.707, figure 2 shows a plot of t_{acq}, phase variance and BER degradation as a function of loop bandwidth B_L. Clearly as the loop becomes narrower to achieve lower BER degradation by having lower σ_θ^2, the acquisition time becomes impractical for typical short TDMA satellite bursts [8]. Meanwhile, if the loop bandwidth is widened to achieve faster acquisition, both phase variance and BER deterioration become large. Ideally and based on our lab experience, a typical Costas loop bandwidth of 25-50 Hz is required for satellite modems with data rates of less than 56 kbps using MPSK. This allows for a maximum phase variance of 10^{-3} rad^2 (or a standard deviation of σ_θ=1.8105°), which results in a BER degradation less than 0.1 dB. One way to speed up feedback loops is by utilizing frequency sweep or variable loop bandwidth, both of which are complex to utilize and may result in destabilizing the loop operation. Loops with a variable loop bandwidth have been utilized [12], however their implementation creates a multiple magnitude of implementation problems, and needs continued maintenance. DFT aided loops have also been used, however a large number of DFT bins (high resolution) is required to estimate the carrier to an offset less than 10 Hz [8], such resolution requires huge computational resource and have rendered useless for mobile low power receivers.

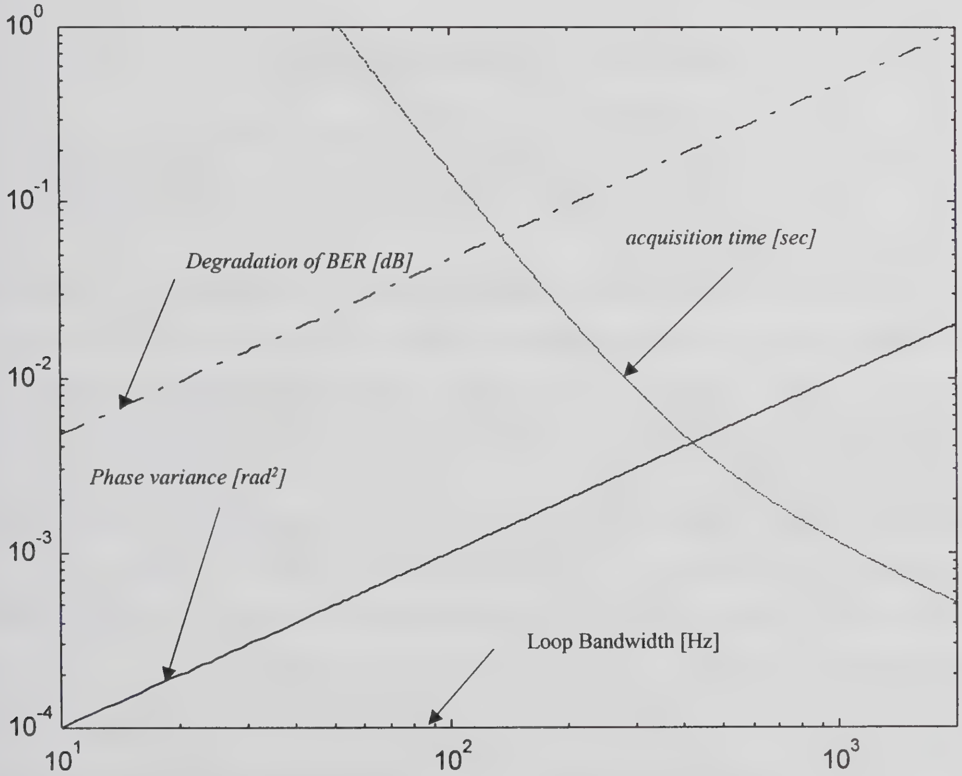


Figure 2: Loop bandwidth versus carrier phase variance and BER deterioration for Feedback type carrier recovery systems.

II.1 Open Loop Mth Power Frequency Acquisition Algorithm

To derive the mathematical representation of this algorithm, we start by representing the received matched filter complex M-PSK baseband signal in AWGN $n(t)$ by

$$z(t) = e^{j(2\pi\Delta ft+\theta(t)+\phi(t))} + n(t) \qquad (4)$$

where $\phi(t)$ is the phase due to the transmitted symbols, $\theta(t)$ is the phase term that includes the carrier phase error and phase rotation introduced by the propagation channel, which may vary in time, however for most of our analysis its assumed to vary slow when compared to the variation in phase due to symbols, Δf is the frequency offset introduced by either channel effects, or offset errors due to the various up and down conversions that the signal encounters while being transmitted and received. To perform frequency offset detection, we pass the matched filter signal through a differential digital detector which eliminates the carrier phase error rotation, by $z(kT)z^*((k-N)T)$, this yields a signal that contains frequency error offsets given by (5)

$$\begin{aligned} z(kT)z^*((k-N)T) &= \exp\{j(2\pi\Delta fkT \\ &-2\pi\Delta f((k-N)T)+\theta(kT) \\ &-\theta((k-N)T)+\phi(kT) \\ &-\phi((k-N)T))+n'(kT)\} \end{aligned} \qquad (5)$$

where k is the index of the baseband samples, $n'(t)$ is a complex AWGN with real and imaginary parts that are independent and have a variance of $\sigma_n^2 = [2E_s/N_0]^{-1}$, or $\sigma_n^2 = T[T_s 2E_s/N_0]^{-1}$, where T_s is the sample time. The differential phase detection causes cancellation of $\theta(t)$, because such phase variation is assumed to be equal between two consecutive symbols, so the component $e^{j(\theta(kT)-\theta((k-N)T))}$ in (5) yields a unity term. For simplicity we will lump the frequency offset terms due to offset difference between N consecutive samples as being $e^{j(2\pi\Delta fkT-2\pi\Delta f((k-N)T))} = e^{j(2\pi\Delta fkT)}$ (notice that we assume *N=1* or one symbol period unless otherwise indicated, or the over-sampling factor, such as denoting every 4th sample is a symbol period). The term $e^{j(\phi(kT)-\phi((k-N)T))}$ is due to symbol modulations, this term can be removed by processing the signal through an M-power non-linearity, where the value M is an integer equal to the modulation symmetry angle (i.e., for QPSK *M=4*, for BPSK *M=2*). QPSK modulation is removed due to the fact that $e^{4j(\pm\frac{\pi}{4}n)} = 1$, *for n=0,1,2,3....* Doing so yields a noisy sine-wave signal in equation 5 with a fundamental frequency that is equal to the frequency offset associated with the received baseband signal. For QPSK, where *M =4*, this is given by (6)

$$y_k = e^{4j(2\pi\Delta fkT)} + n'(kT) \qquad (6)$$

Designating the k^{th} complex sample of the differential baseband signal as being a real part $\mathrm{Re}(z_k) = I_k$ and an imaginary part $\mathrm{Im}(z_k) = Q_k$, making the current matched filter sample $z_k = I_k + jQ_k$, and the previous sample, which is N-samples back in time is $z^*_{k-N} = I_{k-N} - jQ_{k-N}$. Substituting both terms into (5) yields (7)

$$z_k z^*_{k-N} = \underbrace{[I_k I_{k-N} + Q_k Q_{k-N}]}_{\mathrm{Re}\{z_k z^*_{k-N}\}} + j\underbrace{[Q_k I_{k-N} - Q_{k-N} I_k]}_{\mathrm{Im}\{z_k z^*_{k-N}\}} \qquad (7)$$

Equation 7 represents a frequency-offset detector. It can be shown that

$I_{k-N} I_k + jQ_{k-N} Q_k = \cos(2\pi M \Delta f T)$ and $Q_k I_{k-N} - jQ_{k-N} I_k = \sin(2\pi M \Delta f T)$

Both terms are commonly referred to as being the dot and cross product terms, respectively. Expanding both terms further yields an initial frequency acquisition value given by (8)

$$2\pi M \Delta f T = \arg\{(z_k z^*_{k-N})^M \qquad (8)$$

where $\arg\{x\} = \tan^{-1}(x)$. In order to reduce random variations of (8) due to AWGN and phase residuals from data modulations, equation 8 is averaged over L-symbols yielding a final carrier offset estimator algorithm represented by (9)

$$\Delta\hat{f} = \frac{R_s}{2\pi M}\tan^{-1}\left\{\frac{\sum_{k=1}^{L}\mathrm{Im}[(z_k z^*_{k-N})^M]}{\sum_{k=1}^{L}\mathrm{Re}[(z_k z^*_{k-N})^M]}\right\} \quad [\mathrm{Hz}]\ (9)$$

This algorithm has a maximum capture range of $\Delta f \leq \pm R_s/2M$ Hz, or roughly 10% of the transmitted data rate, where $R_s = 1/T$. For instance, using a 10 ksps symbol rate satellite modem, and a QPSK modulated signal (*M=4*), the upper limit of this algorithm is $\Delta f = \pm 2500$ Hz as shown in figure 3. The algorithm's estimate accuracy as a function of variable SNR is shown in figure 4 for detecting a 1 kHz offset. Clearly the algorithm's performance is very accurate for SNR greater than 10 dB.

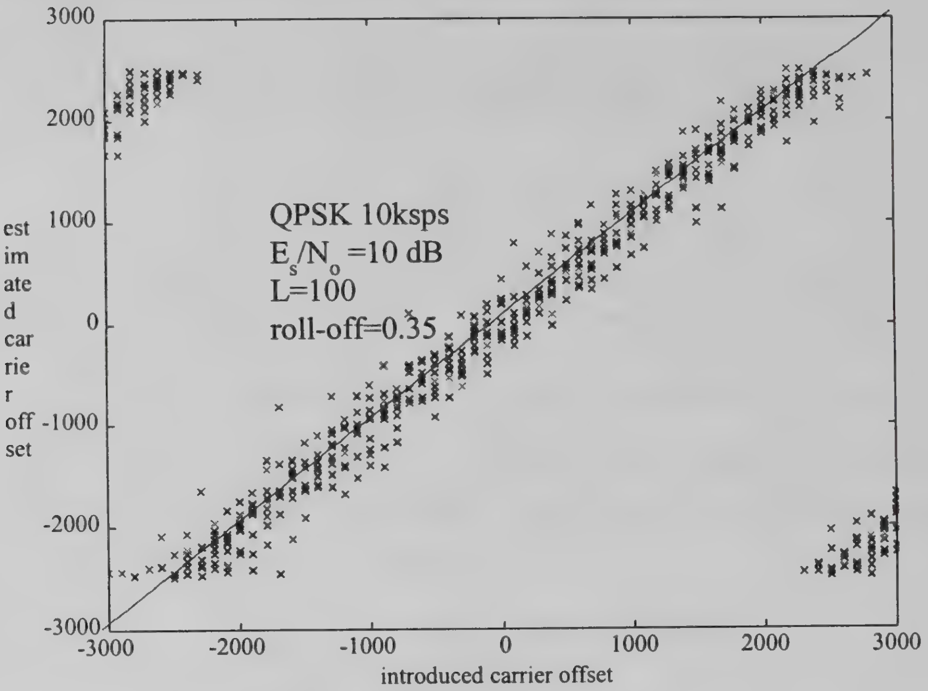


Figure 3: Mth power carrier frequency estimator range performance

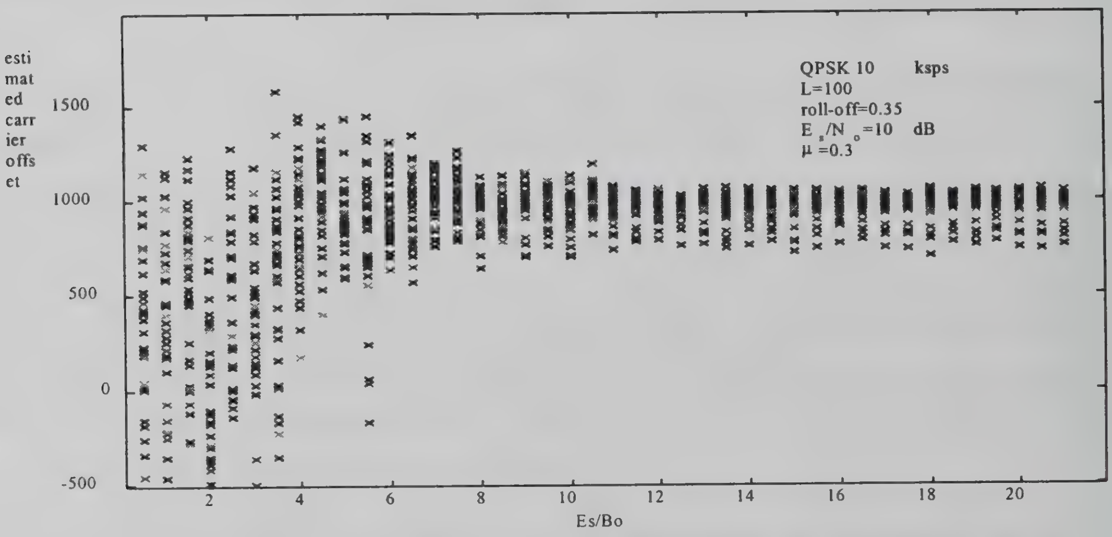


Figure 4: Mth power carrier offset estimator performance for an offset of 1kHz

After frequency offset correction, the matched filter signal still has an arbitrary phase error that have to be estimated and compensated. Before we introduce the phase estimation stage, it is of interest to present several other frequencies offset algorithms that differ from (9) in term of implementation and performance.

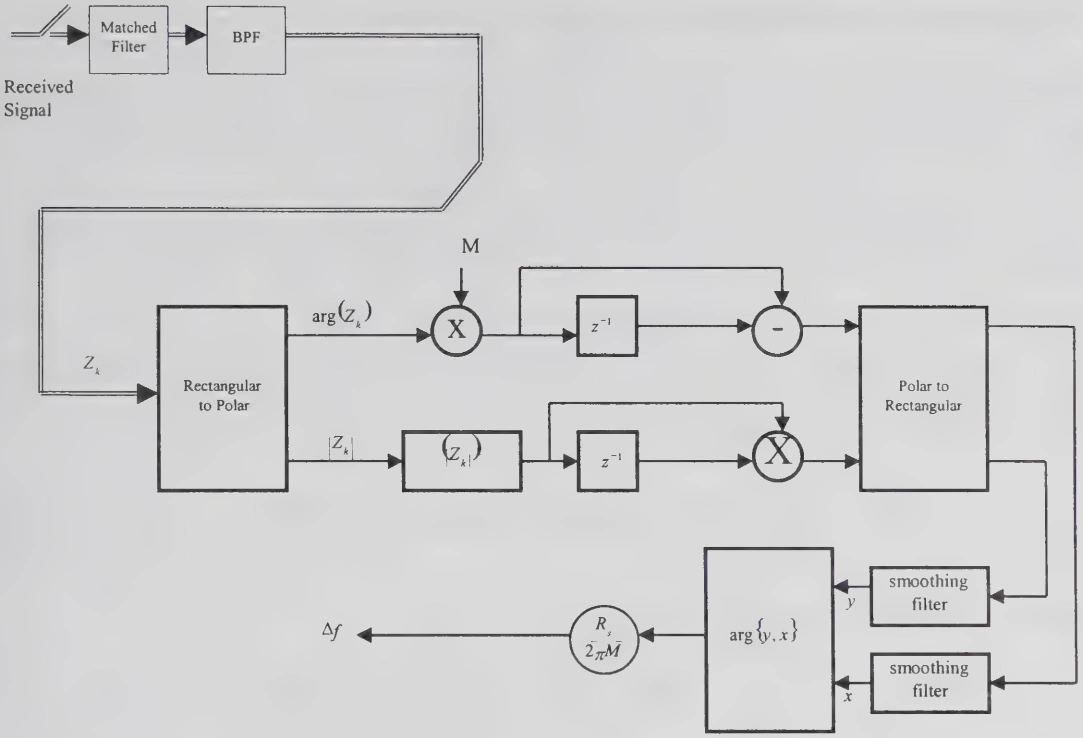


Figure 5: Viterbi carrier offset estimator algorithm

II.2. Viterbi Frequency Acquisition Algorithm

Another form of the M$^{\text{th}}$ power algorithm was developed by Viterbi [6]. This algorithm employes two separate non-linearity operators, one operates on the argument $M\arg\left(z_k z_{k-N}^*\right)$, and one operates on the magnitude of the Matched filter output $\left(\left|z_k z_{k-N}^*\right|\right)^\ell$. This algorithm have been proven to yield better performance for some particular modulations for low signal to noise ratios (especially for QAM modulations [6]). The Viterbi frequency offset estimates is defined by (10)

$$\Delta\hat{f} = \frac{R_s}{2\pi M}\arg\left\{\sum_{0}^{L-1} d_m \left(\left|z_k z_{k-N}^*\right|\right)^\ell e^{jM\left(\arg\left(z_k z_{k-N}^*\right)\right)}\right\} \quad (10)$$

where d_m is a smoothing filter designed to provide robustness when operating at low SNR and to counteract slow signal variations that may be present due to multi-path attenuation. Figure 5 shows an implementation of this algorithm. The smoothing filter can be a simple single low pass filter with a digital transfer function given by $d_m = \frac{1}{1-\mu z^{-1}}$, where μ is a constant such that $0 \le \mu \le 1$.

II.3. Symbol Correlation Based Acquisition Algorithm

Although simpler to implement, the Mth power and Viterbi algorithms presented above suffer from self-noise generated by the use of the non-linearity process for removing data modulations. For QPSK, the use of the 4th order non-linearity results in a 6-dB rise of noise floor, and 3 dB of noise increase for the BPSK signals. Several other algorithms have been employed which are based on the sample to sample correlation function and do not use non-linearity processing. These algorithms involve more computational complexity. The complexity comes from the fact that it takes $N(2L-N-1)/2$ complex multiplications to perform the auto-correlation of two consecutive complex samples of the matched filter. One such algorithm was developed by Luise and Reggiannini in [14]. The frequency-offset estimations is given by (11)

$$\Delta\hat{f} = \frac{R_s}{2\pi}\sum_{k=1}^{N} w(k)[\arg R(k) - \arg R(k-1)] \qquad (11)$$

where the window function is given by

$$w(k) = \frac{3[(L-k)(L-k+1) - N(L-N))]}{N(4N^2 - 6NL + 3L^2 - 1)}$$

and the symbol correlation function given by

$$R(k) = \frac{1}{L-j}\sum_{k=j}^{L-1} z(k)z^*(k-j)$$

where$1 \le j \le N$, $N \le L/2$. The algorithm's capture range is given by $\Delta f = 2R_s/10$ Hz, which is comparable to the Viterbi and the M^{th} power algorithms.

Kay introduced a similar version to this algorithm, where the frequency estimates is given by (12)

$$\Delta\hat{f} = \frac{R_s}{2\pi}\sum_{k=1}^{N} w(k)\arg\{z(k)z^*(k-1)\} \qquad (12)$$

with the window function given by

$$w(n) = \frac{6k}{N(N+1)(2N+1)}$$

Luise&W [10] introduced another form of this algorithm, where the frequency estimate is given by (13)

$$\Delta\hat{f} = \frac{R_s}{2\pi}\arg\left\{\sum_{k=0}^{N-1} w(k)e^{j\arg\{z(k)\} - j\arg\{z(k-1)\}}\right\} \qquad (13)$$

Where the window function (parabolic) is given by

$$w(k) = \frac{6k(L-k)}{L(L^2-1)}.$$

L&W's capture range is $\Delta f = R_s/10$ Hz.

Fitz introduced another version in [10], with frequency estimation is given by (14)

$$\Delta\hat{f} = \frac{2R_s}{\pi N(N+1)} \sum_{k=1}^{N} \arg\{R(k)\} \quad (14)$$

with a capture range of $\Delta f = R_s/2N$ Hz.

Finally, L&R introduced another version that is similar to (12), where the frequency estimate is given by (15)

$$\Delta\hat{f} = \frac{R_s}{\pi(N+1)} \arg\left\{\sum_{k=1}^{N} R(k)\right\} \qquad (15)$$

with a capture range of $\Delta f = R_s/N$ [Hz].

Mengali and Morelli reported simulation [10], which showed that these algorithms have the same performance for SNR more than 14 dB for QPSK modulations. However, with different capture ranges as indicated above.

II.4. DFT-Based Acquisition Algorithm

Another technique for estimating the frequency offset is based on DFT [9]. Unlike the algorithms discussed before; this algorithm does not require symbol-timing estimates. Figure 6 shows the digital implementation of this algorithm. The frequency-offset estimate is by (16)

$$\Delta f = \frac{R_s}{4\pi} \arg\left(\sum_{k-0}^{k=L} [C_{-1}(k) C_{+1}(k)]\right) \text{ [Hz]} \qquad (16)$$

where $C_{-1}(k) = \sum_{L} z(k) z^*(k-N) e^{j\frac{2\pi n}{N}}$, and $C_{+1}(k) = \sum_{L} z(k) z^*(k-N) e^{j\frac{-2\pi n}{N}}$

Notice that $C_{-1}(k)$ and $C_{+1}(k)$ are conjugate pairs, this helps in saving computational resources on the DSP chip. This algorithm has a maximum capture range of $\Delta f \le \frac{R_s}{N}$ Hz, where N is the number of samples per symbols. We have found [10] that this algorithm does not yield acceptable performance, unless the matched filter signal has been filtered using a bandpass filter. The filter bandwidth varies depending on the roll off factor (α)

used. The filter should be designed with center frequency at the spectral component of interest in $z(kT)$, which contribute to the periodic component of interest in $z^*((k-N)T)z(kT)$. Roughly, the BPF should have a center frequency of $R_s/2$ Hz, and a bandwidth that is equal to αR_s.

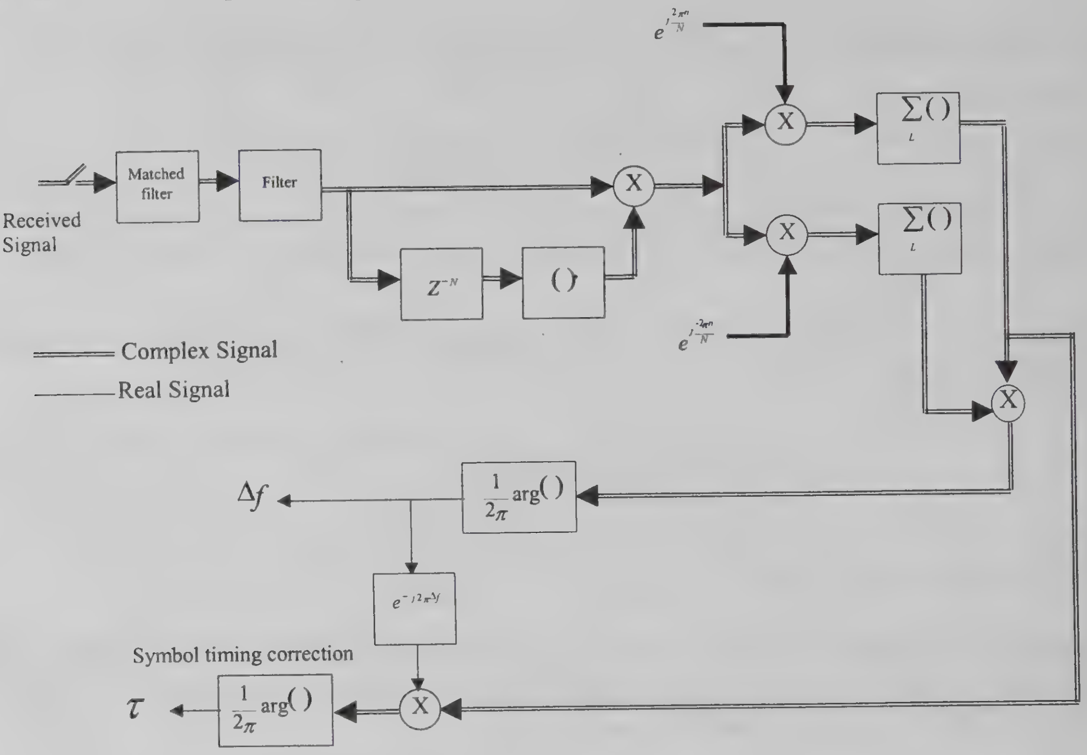


Figure 6: Joint symbol timing and carrier offset estimation algorithm using DFT

III. Conclusion

feedback and feed forward techniques were presented for carrier frequency acquisition. Tradeoffs between these algorithms were introduced in term of their implementation complexity, their estimation accuracy, and their frequency capture range. Computer simulations were used to show their performance in AWGN channels for QPSK modulations.

Acknowledgment

The author would like to acknowledge Dr. Sudhakar of Florida Atlantic University, Dr. Bard and Mr. H. Weeter of Mnemonics Inc. for contributing to this work.

Bibliography

The author hold BSEE and MSEE from University of Colorado. Currently he is a Ph.D. candidate at Florida Atlantic university and principle design engineer at Mnemonics inc., Florida. He can be reached at 321-253-7300 or E-mail mohamed_nezami@hotmial.com.

References

1. J. Bard, M. Nezami, and M Diaz "Data Recovery in Differentially Encoded Quadrature Phase Shift keying", Milcom2000, Los Angeles, CA, Oct 2000.

2. J. M. Nezami and Bard, " Preamble-less carrier recovery in fading channels", Milcom2000, Los Angeles, CA, Oct 2000.

3. M. Nezami and R. Sudhakar, "New schemes for 16-QAM symbol recovery", EUROCOMM 2000 Münich, Germany 17-19 May 2000

4. M. Nezami and R. Sudhakar, " M-QAM digital symbol timing synchronization in flat Rayleigh fading channels", PRMIC, Osaka Japan, Nov-1999.

5. M. Nezami, "DSP algorithms for carrier offset estimation and correction" ICSPAT, Orlando, Florida, USA, November-1999.

6. M. Nezami and H. Otum, "Fine tuning frequency offset errors in M-QAM digital burst receivers using DSP techniques", Third international conference on computational aspects and their applications in electrical engineering, Amman, 19-20 October 1999.

7. M. Nezami, "Non-linear M-QAM digital symbol timing synchronization algorithm suited for wireless handheld radios". Third international conference on computational aspects and their applications in electrical engineering, Amman, 19-20 October 1999.

8. M. Nezami, "Digital synchronization algorithms for wireless burst QAM receivers", Wireless Symposium, San Jose, Ca/USA, Feb-1999.

9. M. Nezami, "An overview of DSP-based synchronization algorithms", Wireless Symposium, MA, USA-Sept-1998.

10. Umberto Mengali and M. Moreli, "Data-Aided Frequency Estimation for Burst Digital Transmission", IEEE trans. Communications, Vol. 45, No. 1, January 1997.

Part II: Frequency Offset Acquisition and Tracking Algorithms

Mohamed K. Nezami
Mnemonics Inc.
3900 Dow Rd
Melbourne, FL 32934

In part I, several algorithms were developed for carrier frequency offset acquisition. The algorithms estimate the offset frequency independent of the carrier phase errors or symbol timing. Phase errors are estimated and corrected, after frequency estimation and correction as shown in figure 1 of part 1. In part II, phase estimation algorithms using open loops and closed loops techniques are presented, a particular application example will be the use of both techniques for acquiring and tracking short TDMA burst satellite signals. Finally, practical methods for implementing these algorithms using commercial DSP chips are introduced.

IV.5: Open Loop Mth Power Carrier Phase Tracking Algorithm

As has been indicated earlier, carrier tracking takes place after frequency offsets have been removed, carrier phase tracking is implemented as shown in figure 1 of part I, where the phase error is estimated by calculating the average rotation of the constellation points. Its is assumed that we are sampling once per symbol at the instants of maximum aye opening. For small residual frequency offset Δf_r resulting from an imperfect frequency offset algorithm performance, the phase variation over one burst is $\theta = 2\pi\Delta f_r LT$, if the observation interval for phase estimation is kept relatively short, this phase variation can be assumed negligible. The carrier phase is estimated by the use of a non-linearity applied to the matched filter complex samples given by (17)

$$s(kT) = F(|z(kT)|)e^{jM\arg(z(kT))} \quad (17)$$

where $F(|z(kT)|) = (|z(kT)|)^{\ell}$ is the non-linearity performed on the magnitude of the samples out of the frequency acquisition and correction algorithm. The value of non-linearity order (ℓ) was investigated by Viterbi. For MPSK modulations, the value of the non-linearity order ℓ can be 4, 8 or 16. Our simulations have shown that $\ell = M$ and equal to 4 was best suited for moderate signal to noise ratios [part I, 6]. Operating on equation 17, the M^{th}-power phase estimates is given by (18)

$$\hat{\theta} = \frac{1}{M}\tan^{-1}\left\{\frac{\sum_{k=0}^{L}\mathrm{Im}[s(kT)]}{\sum_{k=0}^{L}\mathrm{Re}[s(kT)]}\right\} \quad (18)$$

here the range of estimation for this algorithm is limited to $-\frac{\pi}{2M} \le \theta \le \frac{\pi}{2M}$. To make this estimate more robust against fading [3], specially during the presence of fading or residual frequency offsets, the phase estimates (18) are post processed using a low order smoothing filter that will reduce their variance. The post processing filter and the M-power phase estimation algorithm implementation is shown in Figure 7. This post-processing filter can be thought of being a first order Kalman filter smoother [3]. As a result of post processing, the phase is given by (17)

$$\tilde{\theta}_k = \mu\hat{\theta}_k + (1-\mu)\hat{\theta}_{k-1} \qquad (17)$$

where $0 \leq \mu \leq 1$ is the convergence factor. Through simulations, we have found that the value of μ which controls the time constant of the filter, actually can be optimized to yield better performance based on the statistics of fade or the maximum residual frequency offset present [3]. Figure 8 shows the algorithm performance using QPSK signal in AWGN with an intentional phase error. The figure shows an estimation range of $-22.5^0 \leq \theta \leq 22.5^0$.

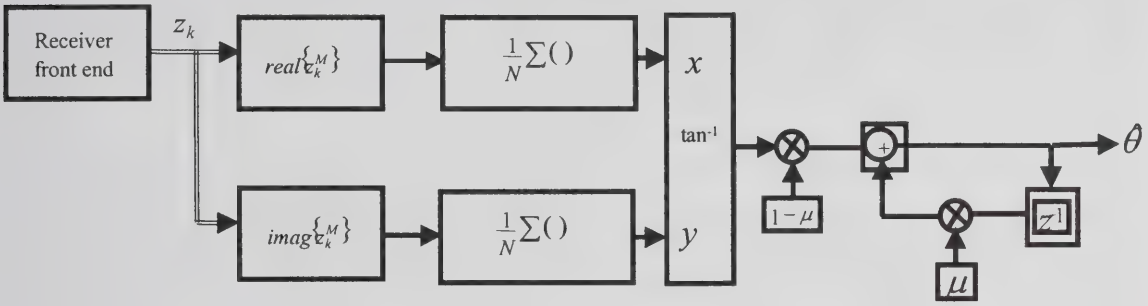


Figure 7: Open Loop Mth power carrier phase estimator

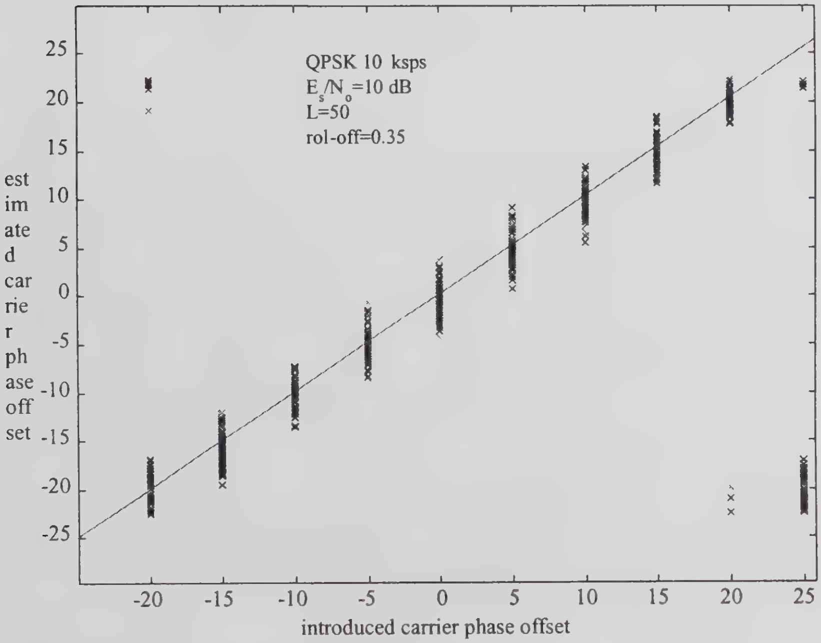


Figure 8: Phase recovery process of Mth power carrier phase algorithm

V. Digital Implementation of Satellite Carrier Acquisition and Tracking

Most satellite systems utilize TDMA, where multiple users share the same channel by using the bandwidth for discrete intervals of time slots. Only one user can access the channel at any instant in time. Time slots are assigned by a network controller for each frame (user). Each terminal has unique phase and frequency offset resulting in unpredictable carrier changes from message to message. To aid acquisition, each message or frame has a preamble or training sequence, which is transmitted as the initial part of each communications burst. The preamble format usually is continuous wave (CW) carrier followed by a dot pattern (alternating or repeating sequence of 1's and 0's). The CW sequence (all 0's) creates a tone that is commonly used by receivers to recover frequency offsets and tracking using some form of feedback loops, such as Costas acquisition and tracking loop. The alternating sequence following the CW part in the burst creates a sine wave at the receiver that has a frequency at half the symbol rate, which is used for symbol synchronization.

Frequency offsets in MobileSat communication terminals are experienced due to several factors, oscillator frequency-uncertainty, oscillators drift, and Doppler effects arising from vehicular motion with respect to the satellite. Depending on the carrier frequency and satellite and ground receiver's relative velocity, such frequency offsets can vary from few hundred Hertz up to several kilohertz. LEO and ICO satellites are located at heights of 10,000 – 20,000 kms above the equator, and have a relative velocity of 1500 m/s operating at 2 GHz, this results in Doppler shifts as high as 10 kHz. GEO satellites are located at height of 36,000 km and operate at frequency of 1.5 GHz or C/KU-band, while the satellite is fixed relative to earth, the relative mobile speed of 100 km/hr creates a Doppler shift of up to 138 Hz for L-band signals. For aircraft with speeds up to 1000 km/hr, the maximum Doppler can be as high as 1800 Hz.

There are several methods employed to estimate such Doppler and carrier drifts, then correct for them. First method is by tuning the reference NCO to an initial known offset, during the first portion of the TDMA, the loop is configured to implement a frequency locked loop by using wide loop bandwidth to be able to pull in large frequency offsets, and frequency acquisition occur within few inverse loop bandwidth values with high probability. After the loop having locked, the loop bandwidth is narrowed to implement a phase lock loop to track out phase errors. A second approach is to use a fixed loop bandwidth, however sweep the frequency oscillator (NCO) of the receiver over the uncertainty region ($\pm\Delta f_{\max}$) at sufficiently low rate so as to enable the narrow loop to lock. A third approach has recently been utilized due to the introduction of high speed, low cost DSPs, is to use DFT aided acquisition [6]. The DFT operation would determine the initial frequency offset in relatively short processing time while the loop is open, then the NCO is programmed to the conjugate of that offset resulting in reducing the total frequency offset to a residual offset which is within the pull-in range of the Costas loop.

V.1 Carrier Acquisition and Tracking using Feedback Loops

There are several ways such loops are implemented. The frequency error can be generated in several methods, one such method is to use the M-power offset estimation algorithm in conjunction with a NCO and a complex multiplier, another method is to use the dot and cross product terms of the differential detector (equation 7, part I). Figure 9 shows such implementation. Here the loop is a closed-loop servo mechanism that uses negative feedback to keep the frequency offset of the NCO equal to the complex conjugate of the incoming offset-baseband carrier frequency. The carrier error loop filter is designed with parameters K_{lead} and K_{lag} such that they minimize phase and offset errors, meanwhile yield minimum acquisition time and a narrow loop bandwidth. Both parameters can be adaptive to change the loop bandwidth, so it is wide during signal acquisition, but narrow during tracking. The frequency error detector used, accommodates both QPSK and BPSK modulations. The Mth-power nonlinearity is sued for phase and frequency tracking when the signal is modulated. If there are no modulations (CW), such as the case at the beginning of most TDMA bursts, the non-linearity can be removed for maximum noise performance.

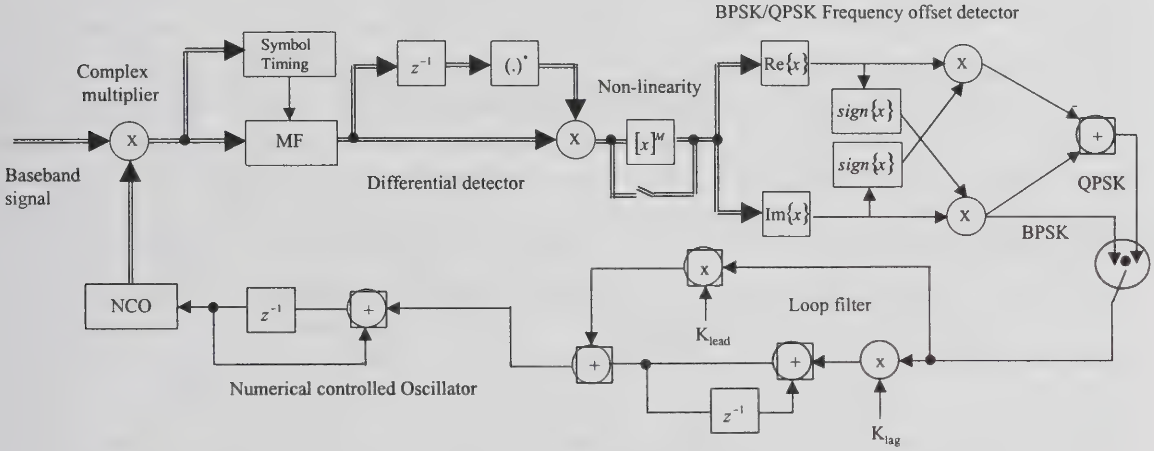


Figure 9: QPSK/BPSK Feedback carrier offset acquisition and tracking loop.

The loop coefficients design starts by using equation 1 in part I, with a given desired t_{acq} and ς, phase detector gain of $K_{phi} = \frac{2^{12}}{\pi}$, and 32 bit NCO again of $K_{NCO} = \frac{2\pi F_{clk}}{2^{32}}$. The loop parameters are given by (20) and (21)

$$K_{lag} = \frac{4\pi f_n \varsigma}{K_{phi} K_{NCO}} \qquad (20)$$

$$K_{lead} = \frac{4\pi^2 f_n^2}{K_{phi} K_{NCO}} \qquad (21)$$

Where f_n is the natural loop frequency.

V.2. DFT-Aided Open Loop Frequency Acquisition and Tracking

To circumvent problems associated with feedback Costas loops in figure 9, such as instabilities, design and implementation difficulties, speed of convergence, and preamble utilization, is to use a modified feed forward open loops by extending their capture range to frequency offsets that are larger than 10% of the data rate. Therefore making them able to cope with typical mobile satellite carrier offsets that are of the order of several kilo-Herts. A modified feed forward techniques proposed here, that does not suffer from hang-ups, and can acquire signals with relatively short time, while using random symbols and not constrained to using a dedicated CW portion of the preamble. Furthermore has an extended capture range beyond the estimation range of conventional feed forward algorithms that are employed previously.

While the digital implementation of conventional Costas loops and feed forward loops are implemented after the matched filter, resulting in limiting the frequency offset acquisition range, as a result, the matched filter attenuates signals with large frequency offsets that lies outside its pass band. The new proposed scheme is implemented prior to the channel-matched filter by using a simple low pass filter that has a pass band equal to $2\Delta f_{\max}$ Hz. The algorithm is primarily developed for carrier acquisition and tracking of signals received from UHF-Military satellites operating with bursts of variable data rates from 1200 to 19200 symbols/s, and a burst duration of 100 to 400 milliseconds. The satellite signal is acquired first at low resolution using relatively short DFT, which brings large frequency offsets to such offsets that are within the capture range of feed forward algorithm, then the feed forward loop fine tunes the small offsets by estimating the carrier offset with variances less than 10^{-6}. Unlike Costas loop, the proposed algorithm does not rely on the use of the CW portion of the burst for initial acquisition and phase tracking, instead it utilizes the alternating sequence which does not have to have synchronized symbols, it only utilizes the spectral contents of the alternating sequence. The initial acquisition is performed based on the simple fact that an alternating sequence pattern yields a line spectra with tones at a frequencies of $f = \Delta f \pm nR_s/2$ Hz, where n is an integer. Figure 10 shows a TDMA QPSK burst with a preamble of alternating sequence running at 40 ksps and 10,000 symbols/s. The figure (bottom) shows the spectral contents of the captured signal, where the presence of two spectral peaks at $f = \Delta f \pm R_s/2$, or 4 and 6 kHz. To demonstrate the proposed algorithm, let's assume a satellite signal that experiences a carrier offset with a range of Δf, where –5kHz $\leq \Delta f \leq$ 5kHz, with $f_s = 40\ kHz,\ R_s = 10{,}000$ symbols/s, and a bin size that is equal to the maximum frequency range of the Feed forward algorithm. Figure 11 shows the performance of the algorithm in tracking the offset carrier of $\Delta f = \pm 5$kHz. The figure shows that by combining DFT and feed forward algorithms in a cascade form, a powerful algorithm become apparent which can overcome many of the problems associated with conventional Costas and Feed forward algorithms used for wide band acquisition and tracking.

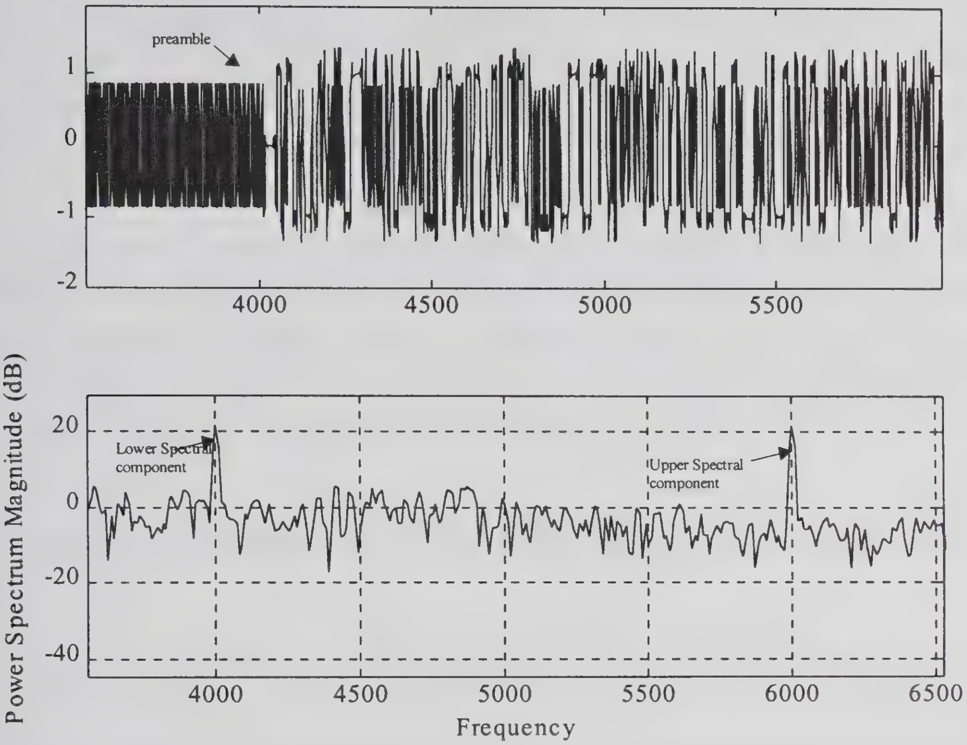


Figure 10: QPSK Satellite TDMA burst.

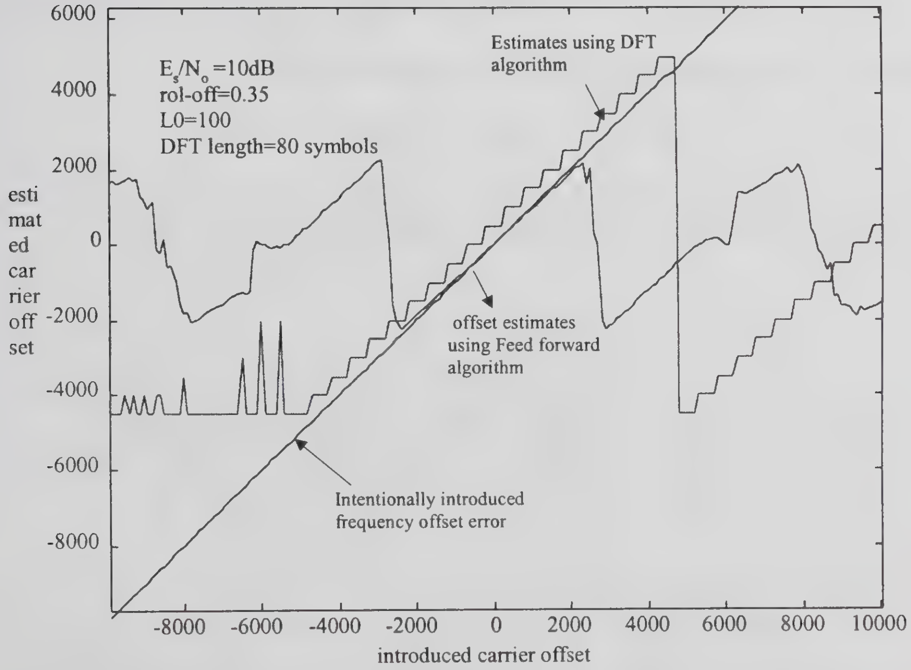


Figure 11: Comparison of feed forward and DFT based frequency offset estimation algorithms

VI. DSP Implementation

Since digital signal processors have no capabilities to do complex mathematical functions, such as $|x|$, $\tan^{-1}(x)$, x/y, and $\sqrt{x}$ functions that are used in the implementation of both frequency offset acquisition and phase tracking algorithm. It is necessary to use approximation to be able to implement such algorithms suing available DSP chips.

1. Magnitude approximation: Magnitude calculation of a complex sample signal is a difficult one since it involves the computation of the square root function which is given by $|I + jQ| = \sqrt{I^2 + Q^2}$. An approximation to this is obtained through use of mathematical expansion given by (22)

$$|I + jQ| = \begin{cases} I + 0.375Q & I > Q \\ Q + 0.375I & else \end{cases} \qquad (22)$$

Figure 12 shows a simulation of this approximation in comparison with the actual function.

2. Division approximation: Division also presents a time consuming and tedious operation for most DSP chips, one way to circumvent this problem is by transforming the division $\frac{x}{y}$ to a multiplication, or $x(1/y)$. An approximation for $1/y$, where $0.5 \le y \le 1$ is given by (23)

$$1/y = 2.5859y^2 - 5.8182y + 4.2424 + \ldots \qquad (23)$$

Using equation (23) the mathematical division becomes (24)

$$\frac{x}{y} = x\{2.5859y^2 - 5.8182y + 4.2424\} \qquad (24)$$

Equation (24) is simpler to implement, since it avoids the use of binary shifting operation that is usually performed when implementing division using DSP chips.

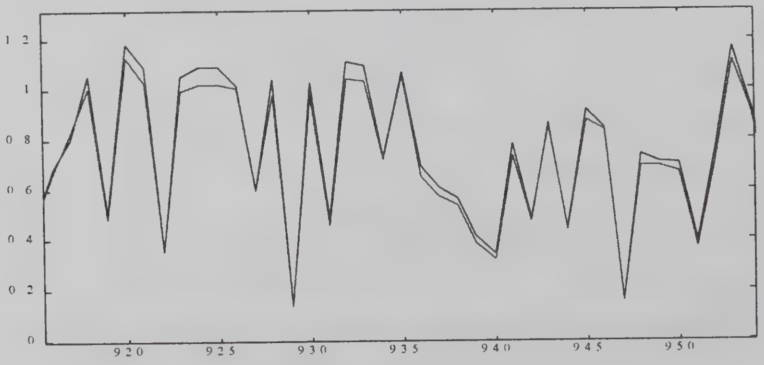


Figure 12: Magnitude approximation algorithm versus actual vector magnitude

3. tan^{-1}(x) Approximation: tan^{-1}(x*)* not only needs table look up to be implemented, but it yields correct answer only in the first two quadrants, or for angles from 0° to 180°. Figure 13 shows an algorithm proposed here that present an approximation to solving the $\tan^{-1}(x)$ function that is implementable using DSP chips. The approximation is given by (25)

$$\arg\{x_k\} = \begin{cases} -\frac{\pi}{4} ratioI(k) + \frac{\pi}{4} & 0^0 \le \theta \le 90^0 \\ -\frac{\pi}{4} ratioII(k) + 3\frac{\pi}{4} & 90^0 \le \theta \le 180^0 \end{cases} \qquad (25)$$

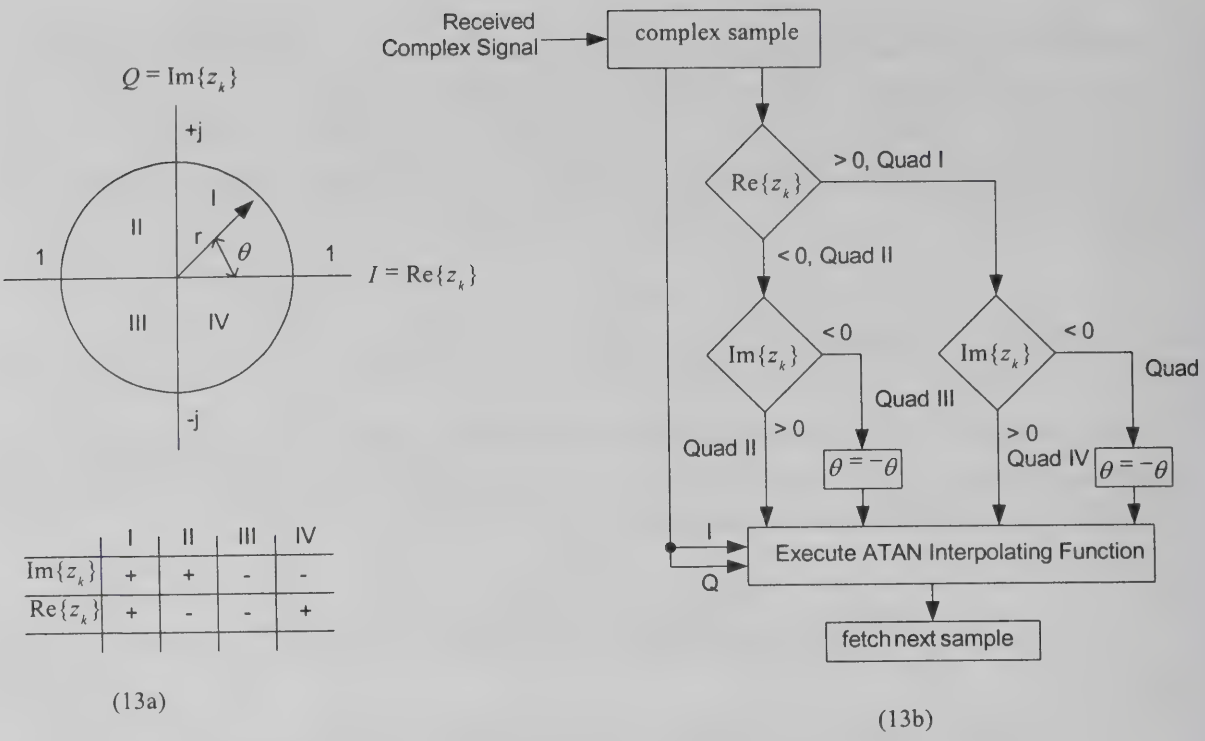
Here x can be $\left[z_k z^*_{k-N}\right]^M$ for the frequency estimation algorithm (9), or $\left[z_k\right]^M$ for the phase tracking algorithm (18). *ratioI (k)* and *ratioII (k)* are a Lagrange quadratic equations obtained by numerical analysis techniques [1] given by (26)

$$ratioI(k) = \frac{\mathrm{Re}\{z_k z^*{}_{k-N}\} - abs[\mathrm{Im}\{z_k z^*{}_{k-N}\}]}{abs[\mathrm{Im}\{z_k z^*{}_{k-N}\}] + \mathrm{Re}\{z_k z^*{}_{k-N})\}} \qquad (26a)$$

and

$$ratioII(k) = \frac{\mathrm{Re}\{z_k z^*{}_{k-N}\} + abs[\mathrm{Im}\{z_k z^*{}_{k-N}\}]}{abs[\mathrm{Im}\{z_k z^*{}_{k-N}\}] - \mathrm{Re}\{z_k z^*{}_{k-N}\}} \qquad (26b)$$

The location of the angle with reference to the quadrant I and II can be found through the sign of $\mathrm{Re}\{z_k z^*{}_{k-N}\}$, while the location of the angle in Quadrant III and IV can be found by checking the sign of $\mathrm{Im}\{z(k)z^*(k-N)\}$ as shown in figure 13a.



	I	II	III	IV
$\text{Im}\{z_k\}$	+	+	-	-
$\text{Re}\{z_k\}$	+	-	-	+

(13a)

(13b)

Figure 13: *arg{x}* DSP estimation algorithm

The algorithm checks the sample's sign of the real part (i.e., *sign{I}),* for negative *I(k),* the angle must be in quadrant II, otherwise its in Quadrant I. The sign of the imaginary part (i.e., *sign{Q}*) indicates whether the angle is in Quadrant III or Quadrant IV, which then inverts the angle value. This is performed such that the computation is only needed to be performed over quadrant I and II. For angles in quadrant III and IV, only inversion is needed. Figure 14 shows the algorithm performance compared to the actual $\tan^{-1}(x)$. Clearly very small error is noticed especially at the interpolation points of 0^0, $\pi/4$, $\pi/2$, $3\pi/4$, and π. If more accuracy is required, a higher order polynomial can be used [1] to compute *ratioI* and *ratioII.* Another attractive feature of this algorithm lies in the fact that amplitude variation in both Q and I, does not effect the angle estimation in (26). This is useful in particular with cases where multi-path fading attenuation is present due to mobile terrain effects.

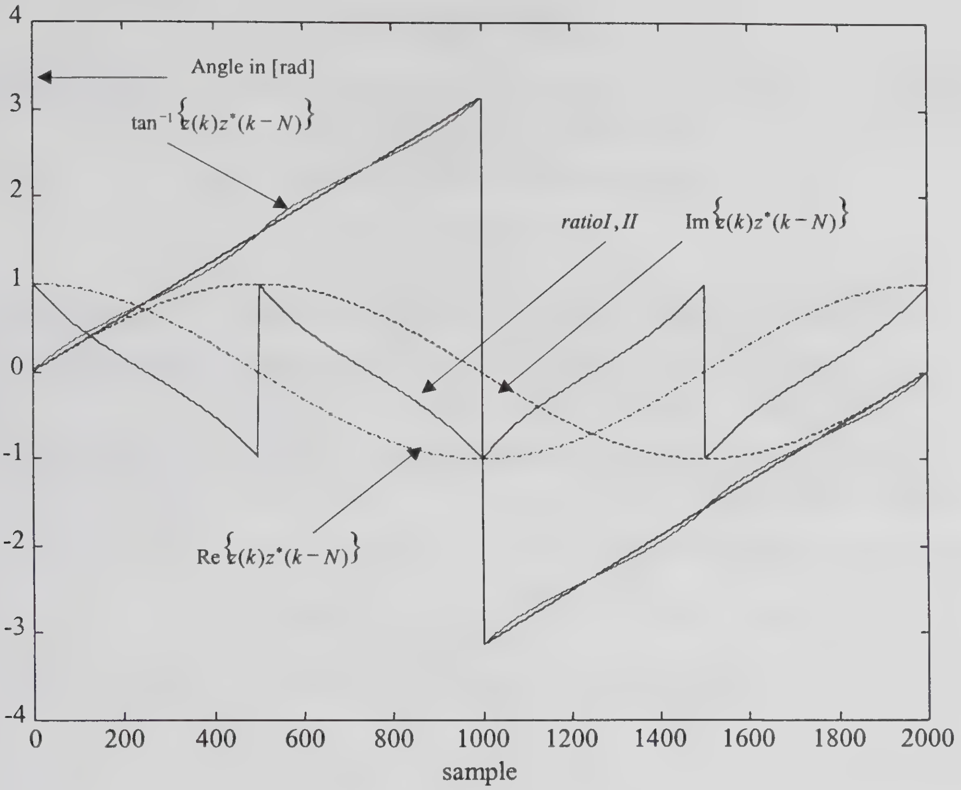


Figure 14: *arg{x}* DSP algorithm performance

The algorithm in (9) in conjunction with (25) was implemented using Texas Instrument C50 processor, with sampling rate of 150 kHz and a matched filter signal of 25 kHz having a frequency offset introduced by the satellite dynamics and of Δf = ±20 kHz. As a result of the frequency offset present, the digital baseband signal had a frequency of 5 kHz to 45 kHz. The algorithm performance was excellent, except that care had to be taken for frequency offsets at the high end of the spectrum, where the oversampling rate was only 3.30, this causes a sample to sample maximum phase variation of $\theta_{\max}(k) = 360^0 f / f_s$, or $\theta = 108^0$ of phase shift, as a result, only 72^0 of phase margin is available, before the sample to sample phase shift introduces ambiguity due to noise spikes while operating at low SNR. Here, often cycle slips will wrap the phase around to the opposite side of the estimation interval, resulting in bad estimates, this was very annoying in the lab, but further post processing and proper filtering resulted in improved receiver performance.

VII. Conclusion

Feedback and feed forward techniques for carrier frequency acquisition and phase tracking algorithms were presented. New algorithms that can acquire signals with large offsets were introduced and characterized. Characterization of these algorithms was conducted using simulations. Practical DSP implementation of these algorithms were introduced and discussed.

Acknowledgment

The author would like to acknowledge Dr. Sudhakar of Florida Atlantic University, Dr. Bard and Mr. H. Weeter of Mnemonics Inc. for contributing to this work.

References

1. Numerical Recipes in C : The Art of Scientific Computing; William H. Press, et al; 1999.

2. J. Bard, M. Nezami, and M Diaz "Data Recovery in Differentially Encoded Quadrature Phase Shift keying", Milcom2000, Los Angeles, CA, Oct. 2000.

3. J. M. Nezami and Bard, " Preamble-less carrier recovery in fading channels", Milcom2000, Los Angeles, CA, Oct. 2000.

4. Henry Helmken, Satellite communication class notes, Florida Atlantic University, 1998.

5. Mnemonics Inc., internal designs review of UHF satellite Modem.

6. B. Shah, S. Hinedi, and J. Holmes, "Comparison of four FFT-Based Frequency Acquisition Techniques", NASA tech. brief Vol. 17, No 10, October 1993.

7. Umberto Mengali and M. Moreli, "Data-Aided Frequency Estimation for Burst Digital Transmission", IEEE trans. Communications, Vol. 45, No. 1, January 1997.

QuakeAPRS

Richard Parry, W9IF

rparry@qualcomm.com
http://w9if.net
http://w9if.net/iweb/quakeaprs/index.shtml

ABSTRACT

QuakeAPRS provides the APRS network with near real-time earthquake information. It is a perl script run as a cron job on a Linux machine. When the script is executed on the hour and half hour, it connects to the USGS to collect earthquake data. It then converts the information to the standard APRS packet format and sends it via the Internet to APRServe. This allows APRS clients to easily display and track earthquakes. QuakeAPRS has been running 24/7 for nearly a year. This paper describes quakeAPRS, the earthquake object format, and lessons learned from the development of this application.

KEYWORDS

APRS, Earthquake, Linux, Packet Radio, Perl, USGS

INTRODUCTION

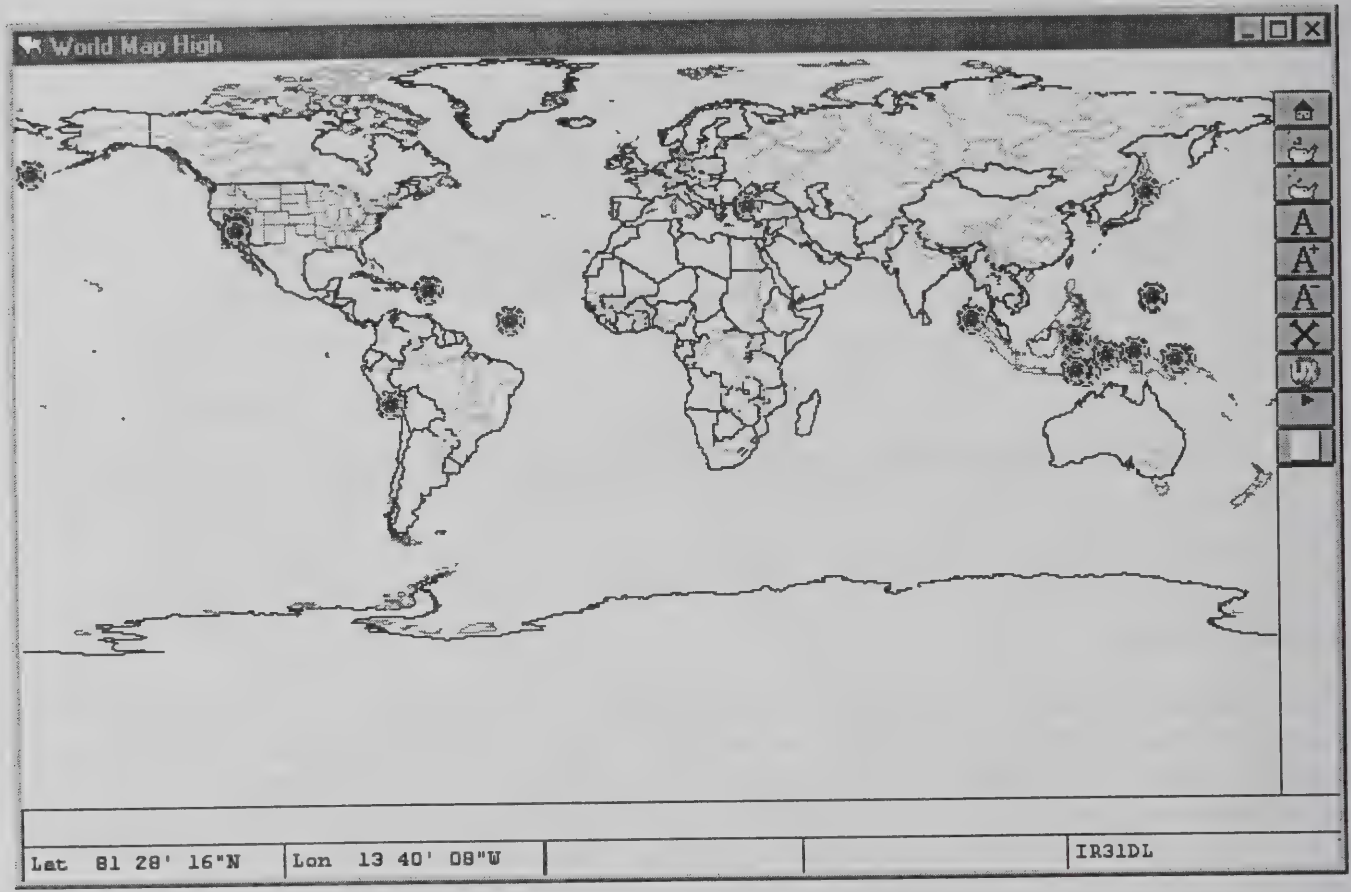
The idea for quakeAPRS started with an email from Bob Bruninga to the Automatic Position Reporting System[1] (APRS) SIG asking for someone to volunteer to write a program that would post earthquake reports to the APRS network. Since the United States Geological Survey (USGS) supplies near real-time earthquake reports on the Internet, all that had to be done was to collect the information, put it in APRS packet format, and send it to APRServe. The original intention was to broadcast the packets to the local APRS RF network where an IGATE would receive the packet and pass it to the Internet. However, the final implementation performs all communication via the Internet, thereby increasing the reliability by removing a single point of failure, the IGATE. This design also has the benefit of allowing confirmation that the packet was received since it communicates directly with APRServe. In the event the connection fails, additional action and notification are possible.

Like most programming projects, the development seemed a simple task, one that could be implemented quickly. However, like most programming projects, once work began, it became apparent that many "what if" scenarios had to be addressed to insure a reliable reporting system. In addition, a standard format for the identification of an earthquake object had to be developed and learning how to interface to APRServe was also necessary to bring the project to fruition.

[1] The APRS formats are provided for use in the amateur radio service. Hams are encouraged to apply the APRS formats in the transmission of position, weather, and status packets. However, APRS is a registered trademark of Bob Bruninga who reserves the ownership of these protocols for exclusive commercial application and for all reception and plotting applications. Other software engineers desiring to include APRS protocols in their software for sale within or outside of the amateur community will require a license from him.

SAMPLE OUTPUT

Figure 1 shows earthquake objects as plotted by WinAPRS. The objects are actual earthquakes with all other APRS objects removed for illustrative purposes. Those who are familiar with earthquake patterns will not be surprised at the distribution of the earthquakes. Japan, Australia, Alaska, California, Turkey and the West Coast of South America are places where earthquakes occur frequently. Due to this wide distribution, users displaying the world map will have the best chance of seeing earthquakes.



Lat 81 28' 16"N | Lon 13 40' 08"W | IR31DL

Figure 1 Earthquake Report on World Map

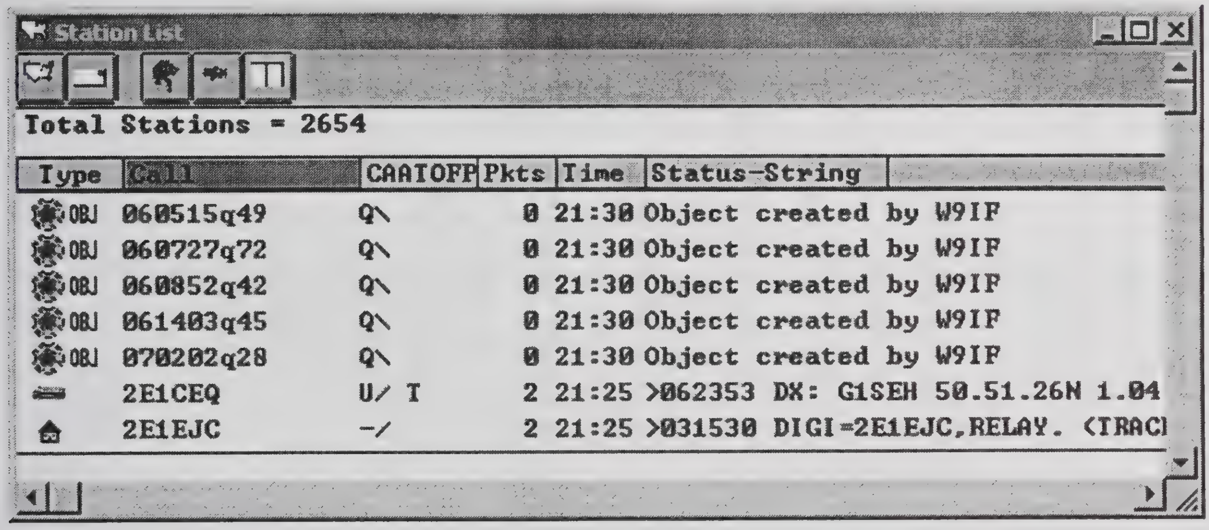
Many earthquakes occur each day, fortunately, most are small. To limit the number of earthquakes to those that are of most interest, two criteria are used by quakeAPRS before the earthquake is considered significant enough to be displayed. First, earthquake objects are limited to those that have a Richter magnitude greater than 3.0. In addition, quakeAPRS sends to APRServe only those earthquakes that are less than 24 hours old. The number of earthquakes reported varies greatly, however, on average, 4 to 10 earthquakes meet this criterion.

QUAKEAPRS FORMAT

Developing an earthquake format for APRS use was an evolutionary process. The first attempt at a format made every object label unique. This is an unacceptable use of APRS objects since it means every 30 minutes approximately 10 new objects are sent to APRServe. Those who leave their APRS clients running continuously would receive several hundred objects daily. This is an unsatisfactory number of earthquake objects, especially since most are duplicates. In addition, these redundant objects represent a significant portion of stations comprising the limited resources of the "stations list".

The final version of the format can be seen in Figure 2, which shows a WinAPRS Station List window. The label for the first object is `060515q49`. The digits to the left of the "q" represent the date and time. In this example, the earthquake occurred on the 6th of the month at 0515z. To the right of the "q" is the magnitude of the quake with the decimal point removed. Therefore, the value of 49 is read as an earthquake with a magnitude of 4.9. It was not possible to include the month and year since the APRS protocol limits the size of the label to 9 characters. However, since by definition all earthquake objects are less that 24 hours old, the year and month are obvious and need not be included. The "q" is used as a delimiter and to make it unique. This means other applications can use a similar format for objects. For example, hurricanes might use the same format by using "h" as the delimiter. The lower case "q" should not be confused with the icon character. The icon information for an earthquake is actually two characters, "Q" and "\".

This format was developed with suggestions from Brent Hildebrand, KH2Z, who uses the same format in APRS+SA.



Total Stations = 2654

Type	Call	CAATOFF	Pkts	Time	Status-String
OBJ	060515q49	Q\	0	21:30	Object created by W9IF
OBJ	060727q72	Q\	0	21:30	Object created by W9IF
OBJ	060852q42	Q\	0	21:30	Object created by W9IF
OBJ	061403q45	Q\	0	21:30	Object created by W9IF
OBJ	070202q28	Q\	0	21:30	Object created by W9IF
	2E1CEQ	U/ T	2	21:25	>062353 DX: G1SEH 50.51.26N 1.04
	2E1EJC	-/	2	21:25	>031530 DIGI=2E1EJC,RELAY. (TRAC

Figure 2 Earthquake Station List

GETTING THE DATA

Numerous web sites are available to obtain earthquake information; below is a small sampling. Some sites show earthquakes by geographical area, some show earthquakes worldwide, while others show only large earthquakes such as those with a Richter magnitude greater than 3.

```
http://www-socal.wr.usgs.gov/recenteqs/Quakes/quakes0.html
http://www-socal.wr.usgs.gov/recenteqs/Maps/Los_Angeles.html
http://www-socal.wr.usgs.gov/recenteqs/Quakes/quakes.big.html
```

Web pages are an excellent method for displaying earthquake data in an easy to understand graphical format. However, the goal of quakeAPRS is to put earthquake data in APRS object format, so we are interested in getting earthquake data in a format that is easy to parse.

A more useful method for obtaining earthquake data is the UNIX "finger" command. It provides a simple and elegant means to obtain the necessary information. The output of "finger" is shown below.

```
[rparry@blue rparry]$ finger quake@gldfs.cr.usgs.gov
[gldfs.cr.usgs.gov]

USGS Central Region Geologic Hazards Team Finger Server.

----
Login name: quake                    In real life: see Ray Buland
Never logged in.
New mail received Sun Aug  6 20:53:17 2000;
  unread since Sat May 27 14:05:59 2000
Plan:
The following near-real-time Earthquake Bulletin is provided by the National
Earthquake Information Service (NEIS) of the U. S. Geological Survey as part of
a cooperative project of the Council of the National Seismic System.  For
a description of the earthquake parameters listed below, the availability of
additional information, and our publication criteria, please finger
qk_info@gldfs.cr.usgs.gov.

This Bulletin is updated every 5 minutes, if necessary.  The same Bulletin
is also available via the Internet at:
http://wwwneic.cr.usgs.gov/neis/bulletin/bulletin.html and that is the
preferred means of obtaining it.
Updated as of Mon Aug 7 03:30:36 GMT 2000.

 DATE-(UTC)-TIME    LAT     LON     DEP   MAG  Q  COMMENTS
yy/mm/dd hh:mm:ss  deg.    deg.     km
00/08/03 13:18:09  34.24N 139.18E  10.0 5.5Mb A  NEAR S. COAST OF HONSHU, JAPAN
00/08/03 13:34:12  39.58N 111.69W   5.8 3.2Ml    <SLC> UTAH
00/08/03 17:01:57  11.88N 143.07E  33.0 5.3Mb B  SOUTH OF MARIANA ISLANDS
00/08/03 19:22:11  17.59S  71.85W  33.0 5.5Ms B  NEAR COAST OF PERU
00/08/03 19:23:38  34.13N 138.99E  10.0 5.2Mb B  NEAR S. COAST OF HONSHU, JAPAN
00/08/03 19:25:55  17.67S  71.97W  33.0 5.2Mb B  NEAR COAST OF PERU
00/08/04 07:17:56  34.27N 139.03E  10.0 4.7Mb A  NEAR S. COAST OF HONSHU, JAPAN
00/08/04 07:47:40   0.01N 126.60E  82.3 5.6Mb A  NORTHERN MOLUCCA SEA
00/08/04 09:18:40  17.65S 178.89W 559.9 4.5Mb B  FIJI ISLANDS REGION
00/08/04 21:13:03  48.85N 142.23E  10.0 7.0Ms A  SAKHALIN ISLAND, RUSSIA
00/08/05 02:55:07   7.28S 128.65E 160.2 5.1Mb A  BANDA SEA
00/08/05 06:13:32  24.41S 112.10W  10.0 5.0Mb B  EASTER ISLAND REGION
00/08/05 06:52:22  48.85N 142.19E  10.0 4.7Mb A  SAKHALIN ISLAND, RUSSIA
00/08/05 08:30:12   6.28S 130.29E 152.0 5.3Mb A  BANDA SEA
00/08/05 19:25:59   5.23S  77.73W  33.0 4.6Mb A  NORTHERN PERU
00/08/05 19:43:09   5.79S 130.40E 181.7 5.3Mb A  BANDA SEA
00/08/06 05:15:40   5.26S  77.58W  33.0 4.9Mb A  NORTHERN PERU
00/08/06 07:27:16  28.84N 139.52E 433.9 7.2Mw A  BONIN ISLANDS, JAPAN REGION
00/08/06 08:52:22  46.23N  75.09W  18.0 4.2Lg    <OTT> SOUTHERN QUEBEC, CANADA
00/08/06 14:03:50  22.00N 142.93E 260.5 4.5Mb A  VOLCANO ISLANDS, JAPAN REGION
00/08/07 02:02:30  40.90N  81.13W   5.0 2.8Lg C  OHIO
```

The date, time, location, magnitude, and brief description are given for each earthquake. QuakeAPRS must parse each of the lines, extract the relevant information, determine if the earthquake meets the magnitude and age criteria, and put the information into APRS packet format.

Below is the same information in APRS format. This data is sent to APRServe for distribution through the Internet. Notice that of the 21 earthquakes reported by the USGS, only 4 meet the age and magnitude criteria.

```
W9IF>APRS:;060515q49*060515z0515.60S\07734.80WQMag 4.9 Depth 33.0 km NORTHERN PERU
W9IF>APRS:;060727q72*060727z2850.40N\13931.20EQMag 7.2 Depth 433.9 km BONIN ISLANDS, JAPAN
W9IF>APRS:;060852q42*060852z4613.80N\07505.40WQMag 4.2 Depth 18.0 km SOUTHERN QUEBEC, CANADA
W9IF>APRS:;061403q45*061403z2200.00N\14255.80EQMag 4.5 Depth 260.5 km VOLCANO ISLANDS, JAPAN
```

QUAKEAPRS INTERNALS

QuakeAPRS is written in perl. It runs on the hour and half hour as a cron job on a Linux machine. It is a relatively simple task to obtain earthquake data, put it in APRS format, and send it to APRServe. However, since it is automated, some means of assuring redundancy and notification in the event of a failure is highly desirable.

Redundancy in obtaining earthquake information is provided by using two sources of data. QuakeAPRS first tries to "finger" the USGS. In the event a finger connection is not possible, quakeAPRS connects to the USGS web page **(http://earthquake.usgs.gov/neis/bulletin/bulletin.html)** and again tries to extract the necessary earthquake data. If either of these connections fail, an email notification is sent to indicate a failure. This allows intervention to determine if there is a problem. Experience shows that the USGS "finger" server is sometimes offline during weekends. However, the USGS web page has been very reliable. This redundancy technique results in near continuous operation.

Getting the data reliably is only half of our goal. The other half is getting the data to APRServe reliably. Since APRServe may itself be offline, a backup system of servers was developed. QuakeAPRS tries connecting to each of the servers below until a connection is made. If any connection fails, an email notification is sent to inform the administrator of a potential problem. If all connections fail, quakeAPRS exits and attempts connection on the next hour or half-hour.

- first.aprs.net
- aprs.socal.interworld.net
- second.aprs.net
- third.aprs.net

Experience thus far shows that although one server may be off line, one or more of the backup servers is typically operational.

CONCLUSION

QuakeAPRS has been running nearly continuously for a year. It started as a simple project to provide a service to the APRS community, but it also turned out to be an interesting and enjoyable learning experience. However, it was not until I received the email below that the seriousness and importance of quakeAPRS was put into perspective.

```
thanks for the quakeAPRS service. i thought i'd let you know that it has
been of great value to us here since the earthquake(s) that hit Nicaragua
a couple of weeks ago. …. thanks rich.

73 de rick, hr2kos
```

ACKNOWLEDGMENTS

Thanks to Bob Bruninga, WA4APR, for permission to use the APRS trademark. Also many thanks to Brent Hildebrand, KH2Z, for suggestions for the final version of the format used by quakeAPRS and APRS+SA.

RESOURCES

1. Bruninga, Bob, "Automatic Packet Reporting System (APRS)," *73*, December 1996, pp. 10-19.
2. Dimse, Steve, "javAPRS: Implementation of the APRS Protocols in Java," *ARRL and TAPR 15th Digital Communications Conference Proceedings*, Seattle, Washington, September 1996, pp. 9-14.
3. Parry, Richard, "Position Reporting with APRS," *QST*, June 1997, pp 60-63.
4. Parry, Richard, "APRS Network Guidelines," *73*, October 1997, pp 19-20.
5. Parry, Richard, "Position Reporting: The Global Positioning System and Linux," *Linux Journal*, July 1998, pp 46-49.
6. Parry, Richard, "XNET: A Graphical Look At Amateur Radio Packet Networks," *ARRL and TAPR 15th Digital Communications Conference Proceedings*, Seattle, Washington, September 1996, pp. 64-75.
7. Parry, Richard, "perlAPRS," *ARRL and TAPR 16th Digital Communications Conference Proceedings*, Baltimore, Maryland, October 1997, pp. 141-148.
8. Parry, Richard, "APRSstat: An APRS Network Analysis Tool" *ARRL and TAPR 17th Digital Communications Conference Proceedings*, Chicago, Illinois, September 1998, pp. 71-76.

Soundmodem on modern Operating Systems

Thomas Sailer, HB9JNX/AE4WA

August 1, 2000

Abstract

Five years ago I presented drivers for using standard PC's with soundcards as packet radio modems [13]. The mainstream CPUs of that era were not quite powerful enough for complex signal processing, so the design at that time had to trade robustness for computational simplicity. Futhermore, operating system preferences have changed since. It is therefore time to rethink the design. In this article, an up to date implementation of an amateur soundcard packet radio driver is presented that features a common source base supporting all major operating systems, and the most common modulation formats.

1 Introduction

Five years ago I presented drivers for using standard PC's with soundcards as packet radio modems [13]. Later, I also presented Linux drivers [11] and a VxD driver for Windows95 [12]. At that time, my development PC was a 66 MHz 80486, and PCI was just born. To keep CPU utilization at an acceptable level in spite of CPU's with slow multiplication, design decisions trading robustness for computational simplicity had to be taken. Today, processors below 500 MHz are getting hard to obtain, and today's CPU's contain fast multipliers, floating point execution units and multimedia instructions

Operating system preferences have also changed. DOS and early Windows versions either did not have soundcard support at all or their support had too much latency to be usable. Today, all common operating systems support low latency soundcard drivers.

The outline of the paper is as follows. In section 2, the architecture of the 1995 drivers is presented together with its problems. Section 3 briefly presents the architecture of the new driver. Section 4 describes installation and usage of the new driver, and Section 5 discusses how faster transmission could be achieved using unmodified handheld transceivers. Section 6 concludes the article.

2 The 1995 Architecture and its problems

2.1 Audio Input/Output

Five years ago, DOS was a widely used operating system for packet radio. DOS did not have any driver support for soundcards, so the packet radio driver had to implement its own soundcard driver. On the other hand, there were only two standard register level soundcard interfaces, namely the ISA SoundBlaster and the WSS (Windows Sound System) register interface. Virtually all soundcards on the market supported either or both of these pseudo standards.

Windows did have sound driver support relatively early, but the original support had long intolerable latencies for packet radio applications. Low latency sound input/output was later added mainly to support games under the name "DirectSound".

Today, PCI has largely displaced the ISA bus, and with it the old register interfaces, since they were tied to the ISA DMA architecture. On the other hand, the importance of DOS faded and all major operating systems feature low latency soundcard drivers.

2.2 Modems

Because of the limited CPU power available five years ago, the modems had to trade performance for computational simplicity. For example, the 9600 Baud FSK modem lacked a receiver filter, resulting in performance degradations from barely noticeable to devastating depending on the receiver used.

Also, the modem code was designed to operate only at a specific bit rate and at a specific sampling rate, limiting the choice of soundcards that could be used.

Because the modem driver included its own soundcard driver, it had to be running in kernel mode. In kernel mode, it is difficult or impossible to use floating point and multimedia instruction sets. The huge differences of the kernel mode environment forced vastly different code bases for the supported operating systems.

3 The new Soundcard Modem Driver Architecture

One important goal of the new soundcard modem driver is supporting all the major operating systems with constrained developer resources. To achieve this, a common source tree for all platforms is employed. Of course certain tasks still require platform specific code, namely the sound driver abstraction (sound driver interfaces vary widely even among UNIX operating systems), and the packet input/output code. The GNU C Compiler [3] was used because it is the defacto standard compiler on many UNIX-like systems, is freely available and can also target 32-bit Windows [9]. The Windows threading primitives differ from the standard POSIX ones used under UNIX and Linux. A small library has been used that implements the standard POSIX threading primitives on top of the proprietary Windows ones [6]. To enable portable GUI applications, the GTK [1] widget library has been used, because it is one of the two defacto standard libraries under Linux and UNIX, and GUI builder tool [2] and a port to Win32 [7] exists.

Perhaps the most user visible new feature is the fact that the modems are now fully parameterizable. I have been often asked to support higher FSK bitrates and the many 2400 baud AFSK modes. All these modes are now supported. To make this possible, the modems must now support arbitrary sampling rates. Of course there is a lower limit to satisfy Nyquist, therefore each modem declares its minimal sampling rate depending on the user chosen parameters, and the driver selects the largest and rounds it to the next rate supported by the soundcard. Once the actual sampling rate of the hardware is known, the modems compute the filter coefficients needed.

Since the modems now support arbitrary sampling rates, it is easy to run multiple modems in parallel. This way multimode access channels such as combined 1k2 AFSK/9k6 FSK can be supported easily. Of course, the transmitters have to be serialized.

In order to simplify the modem code itself, each receiver runs in its own thread. The main interface to the soundcard is the routine `audioread`. The modem specifies the number of samples requested and the time with a 16 bit integer with wrap around. Note that `audioread` does not return if the user requests the driver to stop, and blocks until the requested samples arrive. My modems usually represent the time internally as a 16.16 fixed point fraction of the sample index, and use polyphase FIR filters to interpolate between samples. As the transmitters are serialized anyway, they share a single thread. Again, `audiowrite` blocks until the specified samples are written. Unlike the read case, the write case does not take a time parameter, the samples are assumed to follow the previous ones without gap.

`pktget` and `pktput` request/deliver HDLC encoded bitstreams from/to the HDLC encoder/packet input/output.

With these brief explanations and the help of the existing AFSK and FSK drivers, it should be possible to understand the modem code and to add additional modems to the framework.

4 Installation and Usage

In this section, the installation is described. Source code and precompiled binaries can be downloaded from [10].

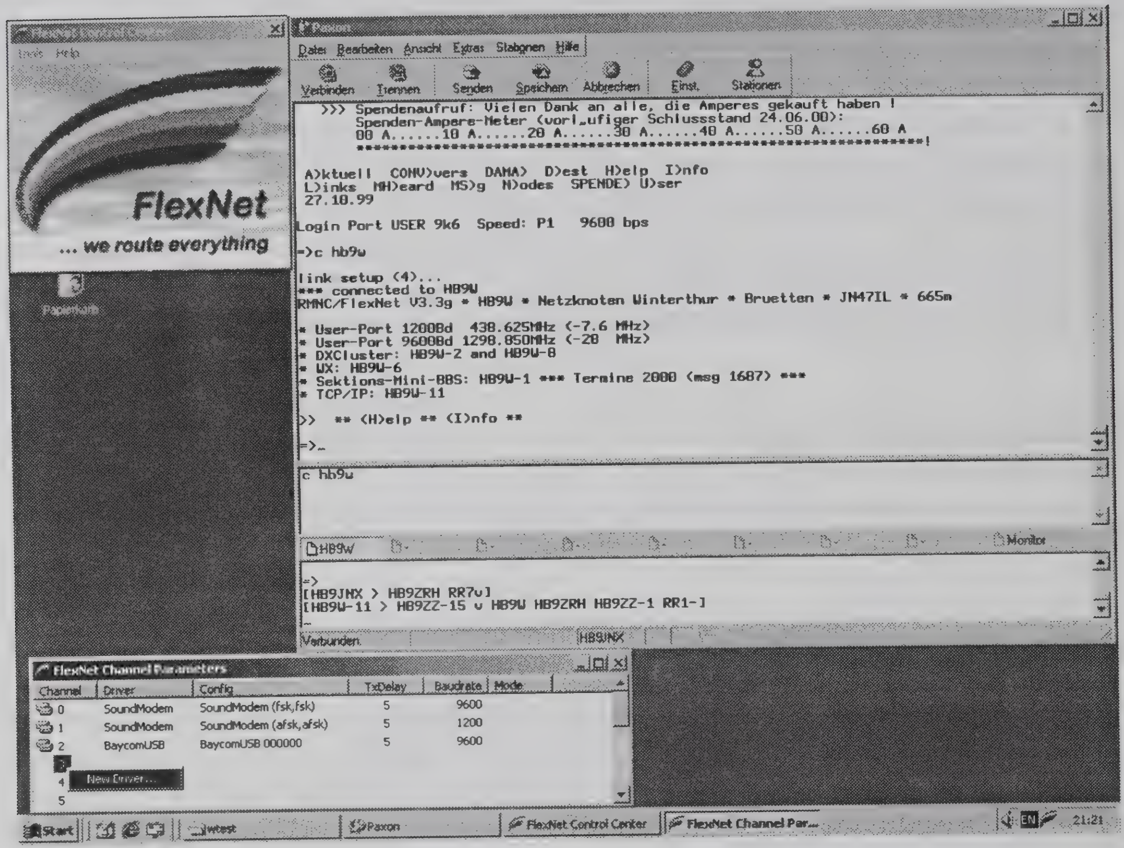
4.1 Windows

Under Windows, FlexNet/32 is used as the software AX.25 stack. FlexNet/32 is freeware for amateur radio use and can be downloaded from [4]. FlexNet/32 supports advanced features, like hop to hop acknowledge and Van Jacobson header compression for TCP/IP. A user friendly terminal program named Paxon and supporting FlexNet's application programming interface is available as well [5]. Figure 1 shows Paxon and FlexNet's channel configuration dialog. Another terminal program supporting the FlexNet API is WPP [14]. It features an user interface similar to well known DOS terminal programs.

Installing FlexNet/32 is easy; unpacking the archive and starting `Flexctl.exe` is all that is needed. Installing the Soundcard Modem driver is simple as well; its archive simply needs to be unpacked into the same directory as FlexNet/32, the new driver gets recognized automatically.

4.2 Linux

Under Linux, there are several options, namely installing the binary RPM which was compiled on RedHat 6.2/i386, or compiling from the source RPM or the source Tarball.



```
   >>> Spendenaufruf: Vielen Dank an alle, die Amperes gekauft haben !
       Spenden-Ampere-Meter (vorl„ufiger Schlussstand 24.06.00):
       00 A......10 A......20 A......30 A......40 A......50 A......60 A
       *****************************************************************!

 A)ktuell  CONV)vers  DAMA)  D)est  H)elp  I)nfo
 L)inks  MH)eard  MS)g  N)odes  SPENDE) U)ser
 27.10.99

Login Port USER 9k6  Speed: P1   9600 bps

=>c hb9w

link setup (4)...
*** connected to HB9W
RMNC/FlexNet V3.3g * HB9W * Netzknoten Winterthur * Bruetten * JN47IL * 665m

* User-Port 1200Bd  438.625MHz (-7.6 MHz)
* User-Port 9600Bd 1298.850MHz (-28  MHz)
* DXCluster: HB9W-2 and HB9W-8
* WX: HB9W-6
* Sektions-Mini-BBS: HB9W-1 *** Termine 2000 (msg 1687) ***
* TCP/IP: HB9W-11

>>  ** (H)elp ** (I)nfo **

=>_
```

```
c hb9w
```

```
=>
[HB9JNX > HB9ZRH RR7v]
[HB9W-11 > HB9ZZ-15 v HB9W HB9ZRH HB9ZZ-1 RR1-]
```

Channel	Driver	Config	TxDelay	Baudrate	Mode
0	SoundModem	SoundModem (fsk,fsk)	5	9600	
1	SoundModem	SoundModem (afsk,afsk)	5	1200	
2	BaycomUSB	BaycomUSB 000000	5	9600	
3					
4					
5					

Figure 1: FlexNet/Paxon

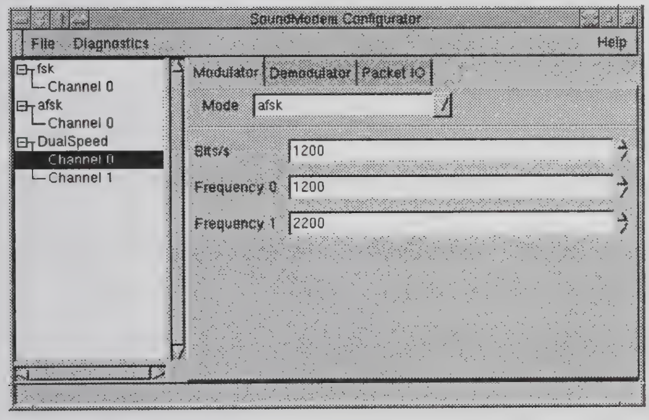
After configuration with the `soundmodemconfig` tool (see below), the modem can be started by running `soundmodem` as `root`. The driver interfaces to the Kernel `mkiss` driver. The result is that the user sees a standard kernel AX.25 network interface that can be used with the kernel AX.25 stack, even though the driver completely runs in userspace. `soundmodem` also sets up the interface callsign and IP addresses.

4.3 UNIX

Under UNIX, there is no kernel AX.25 support, so a usermode AX.25 stack such as xNOS or WAMPES must be used. `soundmodem` exports a KISS stream on a pseudo terminal pair (`pty/tty`), to which xNOS or WAMPES can attach.

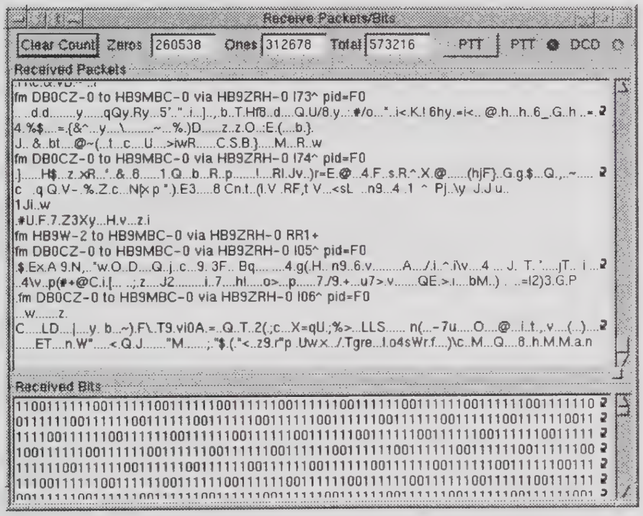
4.4 Configuring the Driver

A graphical application named `soundmodemconfig` makes configuring the driver easy. The tool supports multiple configurations one of which can then be selected at driver runtime. Each configuration supports multiple channels, as depicted in the "DualSpeed" configuration in Figure 2. To ease adjusting the audio levels, `soundmodemconfig` can transmit a test signal and can display the received signal in an oscilloscope like (Figure 5) or spectrum analyzer like (Figure 5) fashion. Figure 3 shows the application monitoring a 9600 baud FSK channel.



Mode: afsk
Bits/s: 1200
Frequency 0: 1200
Frequency 1: 2200

Figure 2: Main Window of Configuration Application



Zeros 260538 Ones 312678 Total 573216

Received Packets

```
fm DB0CZ-0 to HB9MBC-0 via HB9ZRH-0 I73^ pid=F0
.. ..d.d........y......qQy.Ry...5'..".i...]..,.b..T.Hf8..d....Q.U/8.y..:.#/o...".i<.K.! 6hy.=i<.. @.h...h..6_.G..h ..=.
4.%$....=.{&^...y....\.........~...%.)D......z..z.O..:E.(....b.}.
J.. &..bt....@~(...t....c.....U....>iwR......C.S.B.}.....M...R..w
fm DB0CZ-0 to HB9MBC-0 via HB9ZRH-0 I74^ pid=F0
.}......H$...z..xR...'..&..8......1.Q...b...R..p.......!....RI.Jv..)r=E.@...4.F..s.R.^.X.@......(hjF}..G.g.$...Q.,..~.....
c   .q Q.V- .%.Z.c...N|x p ".).E3.....8 Cn.t..(I.V .RF,t V...<sL  ..n9...4 .1  ^  Pj..\y  J.J u..
1Ji..w
.#U.F.7.Z3Xy...H.v...z.i
fm HB9W-2 to HB9MBC-0 via HB9ZRH-0 RR1+
fm DB0CZ-0 to HB9MBC-0 via HB9ZRH-0 I05^ pid=F0
.$.Ex.A 9.N,..'w.O..D....Q..j..c...9. 3F.. Bq.... ....4.g(.H.. n9..6.v.........A..../.i..^.i\v....4 ... J.  T. '.....jT..  i ...
..4\v..p(#+@C.i.[... ..;.z....J2.........i..7....hl.....o>...p......7./9.+...u7>.v.......QE.>.i....bM..)  .  ..=I2)3.G.P
.fm DB0CZ-0 to HB9MBC-0 via HB9ZRH-0 I06^ pid=F0
...w.......z.
C.....LD....|....y. b...~).F\..T9.vi0A.=..Q..T..2(.;c...X=qU.;%>...LLS...... n(...-7u.....O....@...i..t..,.v....(..)....
.......ET....n.W".....<.Q.J......."M......;."$.(."<..z9.r"p .Uw.x.../.Tgre...l.o4sWr.f....)\c..M...Q....8..h.M.M.a.n
```

Received Bits

```
11001111110011111100111111001111110011111100111111001111110011111100111111
01111110011111100111111001111110011111100111111001111110011111100111110011
11110011111100111111001111110011111100111111001111110011111100111110011111
10011111100111111001111110011111100111111001111110011111100111111001111100
11111100111111001111110011111100111111001111110011111100111110011111100111
11100111111001111110011111100111111001111110011111100111111001111110011111
```

Figure 3: Packet Receive Window of Configuration Application

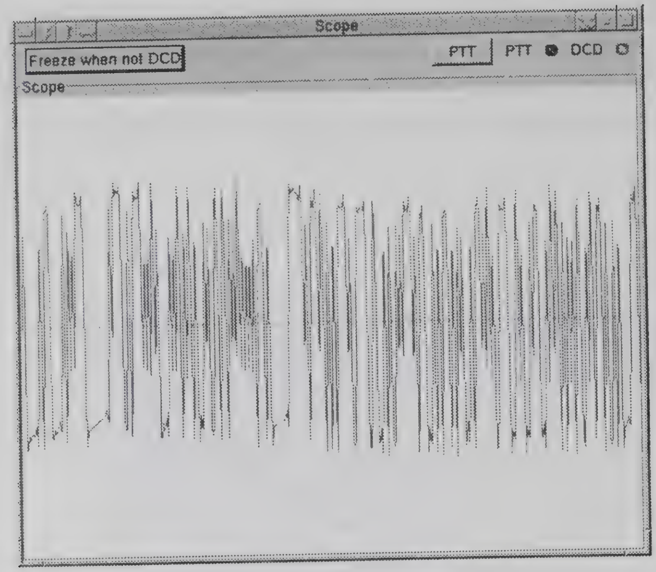
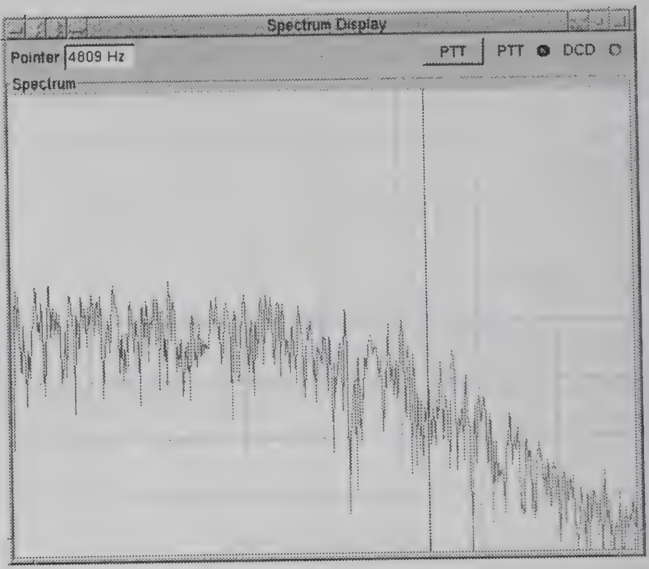


Figure 4: Scope Window of Configuration Application



Pointer 4809 Hz

Figure 5: Spectrum Window of Configuration Application

5 Faster Speeds for Handhelds

9600 Bit/s FSK requires transceivers suitable for this modulation. Handhelds usually do not fall into this category. I have often been asked if it was possible to achieve higher transmission using unmodified handheld radios.

I have been working with handhelds from Standard, namely the C558 and the C701. The reasons to choose these handhelds is that I own them and that handhelds from Standard have the worst frequency response, according to measurements performed by DF9IC [15].

One problem when using handhelds for data transmission is their microphone amplifier AGC, which introduces nonlinear distortions into the signal. To make matters worse, the Standard handhelds I own do not mute the internal microphone when an external signal is connected, thus making it susceptible to ambient noise.

To achieve some immunity to nonlinear distortion, a constant modulus modulation scheme has been used, namely 8PSK at a symbol rate of 2400 baud centered around a carrier of 1800 Hz.

Because the frequency response of the transceiver introduces intersymbol interference, the receiver has to be designed to cope with it. I have used a constraint length 3 viterbi equalizer to combat the ISI. Constraint length 3 captures most of the channel energy, and is about the maximum that can be implemented with reasonable CPU consumption. For every symbol, $8^3 = 512$ trellis branches have to be processed.

Since handhelds are most likely used by end users accessing the backbone network, transmissions will usually be quite short. It is therefore important that the equalizer training is very quick. To achieve this, the modem employs blocks of known symbols. Blocks of 128 data symbols follow blocks of 16 known training symbols. The training symbols are used for timing synchronisation and frequency offset synchronisation. Furthermore, the channel impulse response is estimated using a maximum likelihood estimator [8].

I have not yet decided on what error correcting code to use, the modem transmits raw HDLC encoded bits at the moment.

The modem is in an experimental stage, and tests over the air were successful if the audio levels have been adjusted carefully. Although not ready for deployment, it is included in the source distribution for those who want to experiment. The modem is named “psk”; and it has not yet been converted to cope with variable sampling rates. It requires 9600 Samples/s and therefore won’t work under Windows.

6 Conclusion and further work

Using a standard soundcard and suitable software as a packet radio modem continues to be a viable low cost solution. In this article, the architecture and the usage of such a soundcard modem driver that runs on all major operating systems currently in use has been presented.

Furthermore, the beginnings of a modulation scheme suitable for achieving higher transmission speeds with unmodified handheld transceivers has been presented as well. Clearly, more work will be needed for it to become useful in production use, namely an FEC layer should be added and the modem needs to be converted to support arbitrary sampling rates.

References

[1] The GIMP Toolkit. http://www.gtk.org/.

[2] Damon Chaplin. GLADE – GTK+ User Interface Builder. http://glade.pn.org.

[3] Richard M. Stallman et al. *Using and Porting the GNU Compiler Collection*. Free Software Foundation, 2.95.2 edition, 1999.

[4] PC/FlexNet. http://www.flexnet.home.pages.de/. PC/FlexNet.

[5] Ulf Haueisen and Gerald Schreiber. Paxon. http://www.paxon.de.

[6] Ross Johnson, Ben Elliston, and John Bossom. Pthreads-win32. http://sources.redhat.com/pthreads-win32/, 1999.

[7] Tor Lillqvist. GTK+ and GIMP for Windows. http://user.sgic.fi/ tml/gimp/win32/.

[8] Heinrich Meyr, Marc Moeneclaey, and Stefan A. Fechtel. *Digital Communication Receivers, Synchronization, Channel Estimation and Signal Processing*. John Wiley & Sons, Inc, 1997.

[9] Geoffrey J. Noer. Cygwin: A Free Win32 Porting Layer for UNIX® Applications. http://sources.redhat.com/cygwin/usenix-98/cygwin.html, 1998.

[10] Thomas Sailer. Soundmodem. http://www.ife.ee.ethz.ch/˜sailer/ham/soundmodem.

[11] Thomas Sailer. “cheaper packet” mit Linux. In *12. Internationale Packet-Radio-Tagung*, Darmstadt, 1996.

[12] Thomas Sailer. “PacketBlaster 97” - Soundkarten-PR mit aktuellen Betriebssystemen. In *13. Internationale Packet-Radio-Tagung*, Darmstadt, 1997.

[13] Thomas Sailer, HB9JNX. FlexNet-Workshop. 1995.

[14] Wolfgang Winter. ein Windows Packet Programm (WPP). http://db0exp.de/wpp/.

[15] Wolf-Henning Rech, DF9IC. Equalizer für 1200 Baud. *Adacom Magazin*, (7):pp. 71–73, 1994.

External Common Gateway Interface (CGI) Access to FindU

Thomas M. Schaefer, NY4I
Utah APRS User's Group
11678 Littler Rd
Sandy, UT 84092
ny4i@arrl.net

Abstract

This paper provides programmer documentation on external access to the FindU database. The Common Gateway Interface (CGI) is used as an example of methods for a script or program written on a web server to access that database.

Introduction

When Steve Dimse, K4HG, announced the FindU system at the 1999 APRS National Symposium [1] in Phoenix, little did the APRS community realize the impact this collection of APRS information would have on APRS. Fortunately, Steve has created a site that allows web access to perform certain queries about positions and other APRS information. Of course, being a busy person, Steve certainly cannot program every function the APRS world needs or wants into the system. Fortunately, Steve has allowed external access to the FindU database so inventive programmers can create their own access programs into the FindU system. This relies upon the fact that the FindU system itself is based on the very open, Linux operating system and the MySQL database management system. Using standard routines, it is possible for any programmer with access to the Internet to issue queries to the FindU system and return specific data to satisfy their individual need.

Database Structure

As previously stated, FindU relies upon a MySQL database system running on a Linux system. In order to successfully direct connect to the FindU database, the database structure must be known. Keeping in mind, that FindU is still an evolving system -- meaning the database is subject to change – this is not intended to be an all-inclusive list. It is a current working example at the time of this document's creation. At some point, it is expected that FindU itself will post the actual database format.

The main table for position information is the POSITION table. This table contains several fields that are relevant:

Call	Callsign of the posit
Lon	Long. Of the station
Lat	Lat of the station
Time_rx	Time the spot was received
Speed	speed of the posit
Course	Course from the posit
Raw	Index to the RAW packet table (used to cross-reference the actual packet)

An example SQL query using this data is:

```
SELECT p.call,p.lat,p.lon,p.time_rx,p.speed,p.course,raw.text
from position p, raw
WHERE p.call like 'ny4i%'
AND p.raw = raw.cnt
```

As written, this SQL statement returns all the records in the POSITION table for any callsign starting with "ny4i". Additionally, it returns the actual raw packet from the raw table via an SQL join.

Database Usage

Now that a basic table structure is known, it requires SQL to put it to use. Depending upon your programming language, there are different ways to access that data. The website for MySQL (www.mysql.com) contains a section on programming interfaces to MySQL. Interfaces exist for C++, ODBC (Windows), Java, and DBI (Perl). Since most CGI programs are written in Perl, the focus of this paper is the DBI method. The DBI library is a Perl module that is loaded at the start of a script. As an example, the following code is the Perl method to connect to the FindU database via DBI:

```
$driver = "mysql";
$database = "aprs";
$host = "64.34.101.121";
$user = "guest";
$passwd = "";
$dsn = "DBI:$driver:database=$database:host=$host";
$dbh = DBI->connect($dsn,$user,$passwd);
$drh = DBI->install_driver("mysql");
```

With this code, a connection is now available to access the FindU database directly. Building on the previous example, the method to retrieve some position information using Perl follows:

```
$statement = "SELECT p.call, p.lat, p.lon, p.time_rx, p.speed,
p.course, raw.text from position p, raw WHERE p.call LIKE
\'$call%\' AND p.raw = raw.cnt AND p.time_rx >= $startDate AND
p.time_rx <= $stopDate";

$sth = $dbh->prepare($statement);
$sth->execute;

$numRows = $sth->rows;
```

With this query, the database connection now contains any records available for the requested callsign. Retrieving the individual data fields requires code to loop through the returned database records and utilize the data. An example of this follows:

```
while (my $ref = $sth->fetchrow_arrayref)
   {
      $db_call = $$ref[0];
      $db_lat = $$ref[1];
      $db_lon = $$ref[2];
      $db_time_rx = $$ref[3];
      $db_speed = $$ref[4];
      $db_course = $$ref[5];
      $db_raw = $$ref[6];

      print "<TR>\n";
      print "  <TD VALIGN=\"TOP\">$url</TD>\n";
      print "  <TD VALIGN=\"TOP\">$db_lat</TD>\n";
      print "  <TD VALIGN=\"TOP\">$db_lon</TD>\n";
      print "  <TD VALIGN=\"TOP\">$topoURL</TD>\n";
      print "  <TD VALIGN=\"TOP\">$db_speed</TD>\n";
      print "  <TD VALIGN=\"TOP\">$db_course</TD>\n";
      print "</TR>\n";
   }
```

This code loops through each returned position and populates the "db_" variables. The variables are then used in an HTML table. It is important to keep in mind that while this application displays the data to a web browser, that is not the only use of it. It would be possible to simply search the records to determine maximum speed, or other things for strictly internal reporting.

SQL Cautions

Writing direct SQL queries, while generally easy to learn, does carry a responsibility. It is not terribly difficult to create a Linear Search where every record in the database table must be examined to find a match for your query. To maximize efficiency, indexes are normally used which provide a means to quickly search on a specific field. Similar excessive searches can occur if queries are not dated. In the FindU case, the time_rx field of the positions table should be used to limit the search to a specific data range, when known. By using common sense analysis of the data requested, everyone can successfully use the FindU database.

Windows Access to FindU

While Perl and DBI have been illustrated here, there are other methods to access the FindU database. The author has created Microsoft Excel spreadsheets to access the FindU database via the Microsoft standard, Open Database Connectivity (ODBC). Using ODBC, programs like Word, Access, and Excel have total access to the FindU system. As with the Perl/DBI method, efficient usage is required to ensure responsive queries.

Conclusion

Using commonly available tools, external database access to FindU is available to integrate all APRS data into programs. Using Perl and CGI, it is possible to return all positions of a station. Want to find out where an APRS station has been in the last week, it is now possible. As compared to the original APRServe system, now many different programmers can create queries that suit their particular applications. With this power, does come a responsibility to be a responsible FindU "citizen". But with the proper knowledge of SQL, well-crafted SQL queries can be created to return useful data for the author or the APRS community at large. Of course, without the pioneering efforts of Steve Dimse, K4HG, none of this would have been possible

References

[1] Dimse, Steve, K4HG, "Internet and APRS," [Online Document], 1999 May, Available HTTP: http://www.tapr.org/tapr/ra/dcc99.aprs.4.k4hg.ram,

Server Applications within APRS™ Internet Server Environment

Thomas M. Schaefer, NY4I
Utah APRS User's Group
11678 Littler Rd
Sandy, UT 84092
ny4i@arrl.net

Abstract

This paper discusses programmatic opportunities to access the Internet servers offering APRS data. Particular attention is made to the APRServe system in Miami. Using this system, it is possible for a programmer to create applications that utilize APRS data to create new dynamic and useful applications.

Introduction

With the advent of the APRServe network several years ago, it is now possible to transcend general internet connectivity between APRS programs and create new server applications to utilize this data. This paper presents the background on the server packet architecture as it applies to server access of the data. Through examination of DXSpot, a program to send DX packet cluster spots to the APRS the author illustrates methods to connect to the APRS server network and make use of the data.

Connecting to the APRServe system

The first step to using the APRServe data is getting the APRServe data. Connecting requires some knowledge of socket programming and the port structure of APRServe. APRServe offers data on several different TCP/IP ports. Briefly, TCP/IP ports are different "views" of data on a single system. While a system has a unique TCP/IP address, many different services run on a system. External programs need a way to connect to a distinct service or program. For example, whenever one uses FTP to connect to a server; port 21 is first used to establish the connection. A similar thing happens with a web browser. The web browser generally connects to port 80 to communicate with a specific service (the web server) on the system. In the case of APRServe, the ports that are of general interest are

23	Non-buffered real-time information
10151	Buffered (past 8 hours) of information
14579	Non-buffered Local data (Miami)

For most server applications, buffered data is not useful; therefore, typical server applications connect to port 23. One convenience to port 23 being the non-buffered data is that port 23 is the Telnet port. Therefore, one only need connect to www.aprs.net with telnet to see the real-time feed of data. This single statement is the basis to collect APRServe data in an external application. If the only desire is to read this data, then the programmer simply has the program open a socket to port 23 of the www.aprs.net host, and the data is received as separate messages. The following is a few example packets received from port 23 of the APRServe:

```
KE6QNK-3>APRX46,WIDE3-1:=3759.28N/12200.60W#000/000/
KF4TLJ>APS199/V:>051724z[199YD]*Sheriff's Tactical Amateur Radio
K5QBM-2>APRS,WIDE3-1:!3254.00NN09640.88W#PHG7380/WIDEn-n
KA7PBI-10>APZ034,TCPIP*:=4734.07N/12208.06W-PHG5630/XASTIR-Linux
WB0SBH*>APRS,WIDE,WIDE:!4354.88N/09229.78WoPHG5230 Rochester
N0CALL>APRS,TCPXX*:!3853.00NN010140.88W#PHG1234
```

As these packets are received, whatever parsing is required should be performed and the required action taken. One possible application is searching for messages to a particular callsign. The processing, of course, is dictated by the application need. While the above messages may appear generic, the number of different APRS messages is large. The APRS protocol reference [1] is best referenced to determine all the possible messages. To keep things manageable, it is only required to search for the messages that are of interest to a program. For example, if a program were written to check for callsign requests, that message would typically be in an APRS message. Since an APRS message has a distinct format, only packets following the message format need to be parsed beyond general identification.

Authenticated packets

APRServe contains a mechanism to verify that a message on the Internet has come from an amateur radio operator. All registered APRS programs and anyone else may contain a registration code that allows generation of a password to send messages to the APRServe and then to a local 2m packet network. The idea being that only "verified" packets should be sent to RF, or "gated". The distinction in the APRServe between verified and non-verified packets is the TCP field in the path. By observation of the above examples, it can be seen that KA7PBI-10 sent a position via the Internet. This is indicated by the APZ034,TCPIP string in the packet path. By comparison, KE6QNK-3 was received from 2m via an IGate. This is indicated because the path is APRX46,WIDE3-1 (or any other combination typically found in a X.25 path header). The method used by APRServe to indicate unverified packets is to change the TCPIP path to TCPXX. This is illustrated above by N0CALL. If a server application receives a packet with TCPXX, it is important that this packet not be sent to RF since this is not proven to be an amateur originated packet. FCC rules require that only packets originated by hams be sent over RF.

Sending Packets to APRServe

The other side of the APRServe system is to be able to send packets to the server. The topology of the system states that any packet sent to the system will then be sent to every Internet Gateway (or custom server application) connected to the system. It is this system that allows custom server applications to communicate with the APRS stations over RF. The protocol to send to APRServe is more involved than simply receiving data. First in order to send data that is then sent to RF, the server application needs to be authenticated. To authenticate any server program, the following sting must be sent to the connected APRServe socket:

```
user <callsign> PASS <password> vers <Program Name and version>
```

The APRServe system requires that a password derived from the callsign be obtained to send messages to the RF side of the system.

To obtain a password, send an APRS message to ICQServe with PASSWORD as the message.

Once the login is completed, the application is ready to send messages to the APRServe for possible transmission to the 2m network. The format of the packets is very similar to the APRS "on-air" packets. The only exception is that the path used in the packet is TCPIP*. For example, presume a system call DXSpot wants to send a position packet. The format of the packet is as follows:

```
DXSPOT>APRS,TCPIP*:*:!4028.00NN11151.88W#
```

When this packet is received, it is sent to all the other servers on the system. In a similar fashion, if a server called EMAIL wants to send a message to NY4I, the packet would appear as follows:

```
EMAIL>APRS,TCPIP*::NY4I      :Your email has been sent
```

An interesting feature is that authentication is determined on a port basis. In other words, if the proper login string is sent on the connected socket, any packets sent on that port are considered to be authenticated. It is perfectly reasonable to login with a password of a callsign, and then send the packets from an application name like ICQServe, DXSpot, or EMAIL.

Case Study

To properly illustrate the capabilities of the APRServe system with server applications, an example program will now be examined. DXSpot is a Perl program that connects to both a DX Cluster and APRServe all via the Internet. DXSpot's purpose is to send DX Cluster information over a local 2m network. Regardless of one's specific feelings over the purpose of send DX information on an APRS frequency, there are some valid reasons. Several amateurs have both an APRS radio like the Kenwood D700 in their vehicle as well as an HF rig like the Yaesu FT100 or the Icom 706. The Kenwood radios offer the ability to receive and display DX information sent from a packet cluster. In almost all cases, this information is sent on a frequency separate from 144.39 MHz. If the APRS radio is operating on the APRS frequency, then the packet spots cannot be obtained from the network at the same time. In order to make DX spots more useable for radios monitoring 144.390, a program called DXSpot was created. This program is written specifically to a certain DX Cluster system (CLX) [2]. The CLX packet cluster is a popular Linux-based system that offers Telnet access to obtain DX spot information. While this program's login may differ from other clusters, it is normally only by slight differences. After login to the cluster, the DXSpot program waits for a DX Spot to be received on the cluster socket. After receiving the packet, the spot is then converted into an APRS DX Spot format [3]. The most important part of this program is the following: the originator of the packet is sent to SLCDX (SLCDX because that is where the program runs). The following is an example of a properly formatted DX spot to be sent on the APRServe socket:

```
SLCDX>DXSPOT,TCPIP*:<DX Spot as received from DX Cluster>
```

This packet is then sent to the APRServe. When the APRServe receives this packet, it then sends it out to all the connected IGates. When the IGate in Salt Lake City receives it, it is sent out to the local 2m RF network. This little bit of magic happens because the IGate in Salt Lake City (running aprsd) will send any TCP/IP packet from SLCDX automatically to the RF network. To duplicate this system for ones own area, first find out the exact method to telnet to the local DX cluster (if possible), then modify DXSpot to connect properly. Ask the IGate sysop to add some code to the gate to send packets from the DXSpot software to RF immediately. Airport codes seem to be the adopted standard in the aprsd case.

Future Issues

With the current availability of the FindU APRS data storage system, a fundamental design architectural question now exists. If the server required by an application is real-time, the connecting to the feed seems to make more sense. If the application only requires query on a specific packet, then a database system like FindU is warranted. The application programmer should carefully consider what type of data is required.

Conclusion

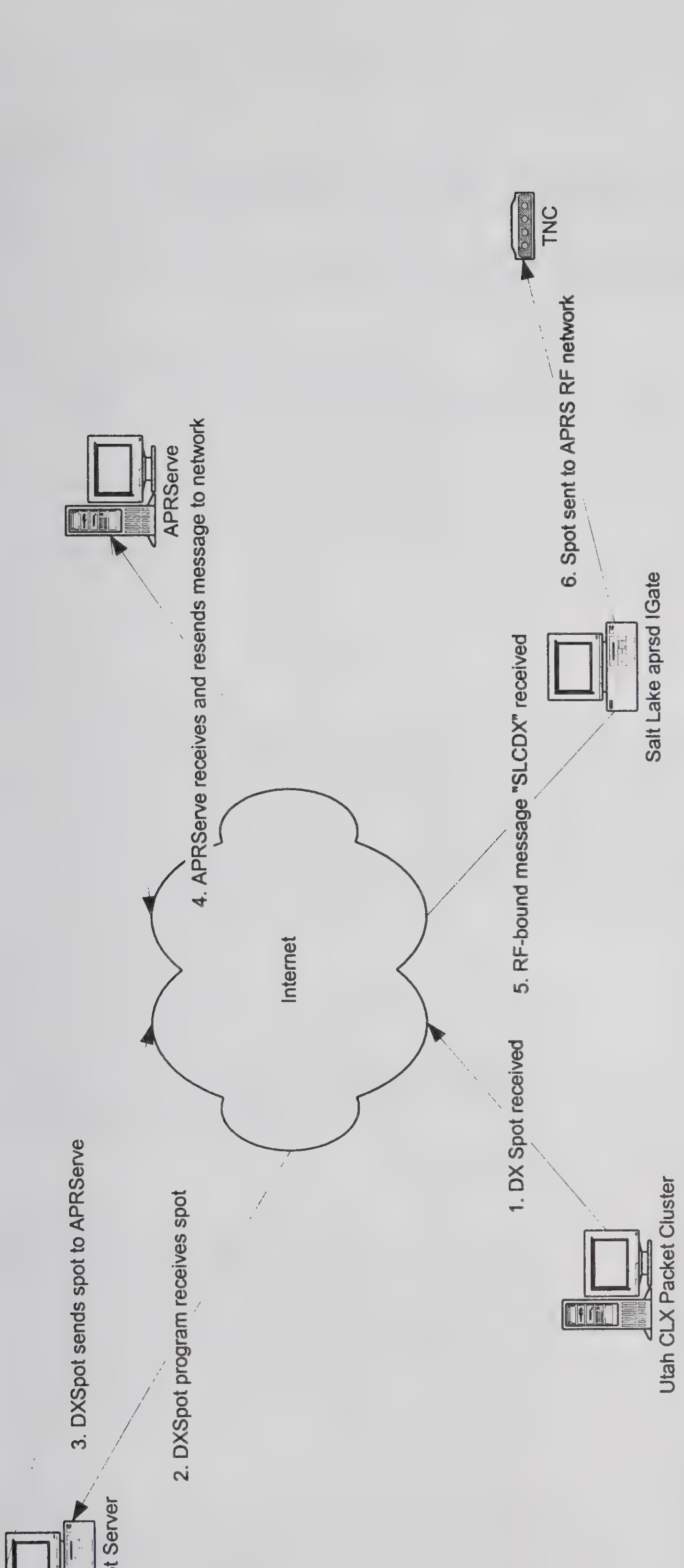
Using existing TCP/IP network programming concepts, it is possible to create useful applications using the APRServe data feed. At the core of the system is a server that allows multiple connections and distribution of data to many different connected programs. The APRServe system is the common link that most local APRS networks share. With this sharing of data, applications such as DXSpot, Email, and ICQServe can be created to make real-time use of the entire worldwide APRS network. With simply a computer and an Internet connection, packets can be negotiated all over the world for the benefit of all APRS users.

References

[1] APRS Working Group, "APRS Protocol Reference," [Online document], 2000 May 1 (Rev 101m), Available FTP: ftp://ftp.tapr.org/aprssig/aprsspec/spec/aprs101m/APRS101m.pdf

[2] Franta Bendl, DJØZY and Bernhard (Ben) Büttner, DL6RAI, “CLX Homepage,” [Online Document], 2000 June 18 (Rev 5.03a), Available HTTP: http://www.clx.muc.de

[3] Ian Maude, G0VGS, “CLX User Manual,” [Online Document], 1999 Dec (Rev 4.03), Available HTTP: http://www.clx.muc.de/user/english/html/userman-4.html#ss4.2


DX Spot Server
3. DXSpot sends spot to APRServe
2. DXSpot program receives spot
Utah CLX Packet Cluster
1. DX Spot received
Internet
4. APRServe receives and resends message to network
APRServe
5. RF-bound message "SLCDX" received
Salt Lake aprsd IGate
6. Spot sent to APRS RF network
TNC

Intelligent Filtering of the APRS Internet Gateway Data Streams

Darryl Smith, B.E., VK2TDS
PO ox 169
Ingleburn NSW 1890
Australia
Phone: +61 412 929 634
Darryl@radio-active.net.au

ABSTRACT: APRS is a fast growing mode throughout the Amateur Radio world, thanks to its combination of computers, radio, packet and the Internet. Such popularity has increased the amount of data on the APRS network to a point where many users, especially those outside North America, are overwhelmed by the volume, or in an attempt to reduce the traffic volume do not enjoy the full benefits if Internet Connected APRS. This paper describes work by the Author into filtering the data stream intelligently to create manageable local networks.

Introduction

In the past 3 years since Internet gating of APRS packets was started, APRS.NET traffic volumes have increased dramatically. Although I have been unable to find any documentation on past traffic volumes, measurements indicate that a complete stream from APRS.NET in mid-2000 would consume more than 25-50 Mbytes per day.

To gate this volume of traffic to RF would require probably 10 times the transmission rate as provided by the current 1200 bps packet modems given the highly distributed nature of APRS and ALOHA networking.

Even the most basic data stream, consisting of only message and related position reports consumes upwards of 4 Mbytes per day, placing a significant load on any transmitting equipment with little benefit to those receiving the transmissions.

The number of stations world wide is beginning to take its toll on the APRS software. We have recently seen instances where the number of stations on the network caused some APRS software to fail. Redraw times have increased so that only the fastest computers can realistically be used to use APRS.

AFilter for instance was written so that end users could filter their own data streams, improving the performance and stability of APRS applications.

Review of Traffic Volumes

Analyzing the APRS.NET traffic stream for about 15 minutes reveals some interesting results. The USA accounts for 79.8% of the packets in this period that I logged the APRS.NET Server. Adding Canada increased the North American traffic to 89.2%.

Country	Packets	%
USA	786	79.8%
Canada	92	9.3%
Netherlands	23	2.3%
Australia	16	1.6%
Switzerland	14	1.4%
UK	11	1.1%
Brazil	8	0.8%
Hungary	7	0.7%
Portugal	6	0.6%
New Zealand	5	0.5%
Honduras	5	0.5%
Germany	5	0.5%
Italy	4	0.4%
South Korea	3	0.3%

Unfortunately, most of the USA based traffic is of little use for the rest of the world. In fact most of the USA traffic is of little use to most other USA based amateurs. By eliminating most of this data stream, the volume of data is far more manageable, and scalable.

In some areas the current trend is to place the entire APRS data stream onto RF. Whilst this is a noble endeavor, it does tend to cause some problems when the data stream contains more information than the channel can support.

Data Segregation

The solution to the data overload problem is to somehow reduce the volume of data that individual users are required to deal with.

Several methods can be used to segregate traffic

- Originating Callsign (Region)
- Position in the world
- Type (Message, Position, etc)
- Redundancy of data

A combination of filters based on these conditions will yield a reduced flow of information, without decreasing the relevant information. The APRSd software already has filters to remove redundant data. Apart from these it has very limited filtering capabilities.

Of course, MIC-E Emergency Packets should be passed unconditionally.

Hierarchical Networks

Anyone who has researched the history of the Internet will know that in the beginning it was a very flat network, with flat protocols to support the network. Even the naming of machines was done on a single-level structure.

As the number of users grew, the old structures were no longer able to support the increased usage. Therefore the internet has been transformed into a highly distributed hierarchical structure.

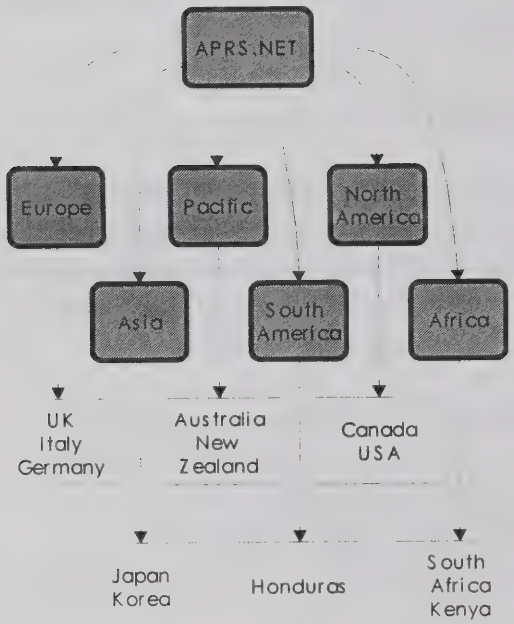
The APRS IGATE system has similarly grown, and is becoming increasingly fragmented and hierarchical. It is growing thanks to the proliferation of Open Source software, as well as easier to use APRS software allowing an IGATE to be setup without actually knowing what you are doing, at almost zero cost.

The haphazard connections between IGATES has caused some problems with APRS messaging. Whilst packets are getting from the extremities of the network to the main APRS server, messages sent to the stations providing these positions will often not get through.

The cause is due to the fact that many of the IGATE systems only feed data into the network, and do not retrieve data because of the sheer volume of data.

Proposed Hierarchy Of Servers

I am proposing that APRS IGATE data be filtered by the region, initially by callsign. Once this happens, filtering by region can be added. Those that prefer the present situation where they can see all the world wide stations will still be able to connect to APRS.NET.



In this proposal, there would be a root APRS server, and a number of regional servers. These regional servers would be permanently connected to the root server exchanging data much as happens with IGATES today.

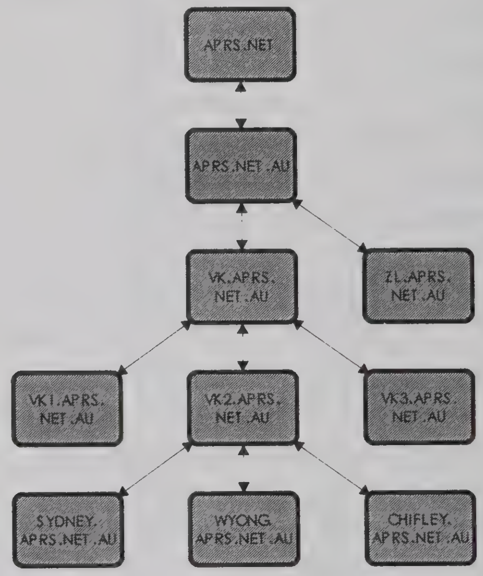
Please note that I have not tried to include any discussions about backup servers in this paper. They are able to operate in a manner similar to the present system with *second.aprs.net*.

Users would be encouraged to connect to a regional server, but there would be no requirement for this. It is anticipated that the APRS.NET root server would become the defacto USA server under this plan.

As you go further down the chain you would see a structure similar to the one appearing on the next page.

This does not however have any advantage over the current structure of IGATE's except the structure of naming. Something else is required before this will give gains required for the whole IGATE system.

To obtain any improvements some form of filtering is required between the levels of the network.



Filtering of Data By Callsign

Australia and New Zealand have some particular properties that both help and hinder the development of our national APRS networks

Australia has a very structured callsign system by state. The Number following the VK in Australian callsigns indicate the STATE that the Amateur lives in. It is rare to find an amateur operating from a state other than what his callsign would indicate, given the size of the each state.

New Zealand has a similar structure, although because of it's size there is little use breaking down traffic much further.

Visitors to Australia must obtain a LOCAL callsign before they may legally operate in Australia, which makes the structure even more viable. However much of the rest of the world does not have such stringent requirements, so another solution is required.

Filtering of Data by Position

Because of the structure of the Australian and New Zealand callsigns I have only recently attempted to filter based on positions. This is the logical next step so that users with non-local callsigns are able to automatically use the local IGATE system.

Development of software to filter the data stream by position has just been completed. The filtering technique being developed is based in Richard Parry's perlAPRS software. It takes a list of valid grid squares, and then uses these to validate the position.

Rules – Filtering Philosophy

Child to Parent IGATE

A number of basic rules can be applied to filter data between the IGATE and the parent IGATE. Any packets that do not satisfy one of these rules will be discarded.

1. All packets from VK*
2. All packets from stations inside VK
3. All messages to VK* or stations inside VK
4. All packets from a station who's packets are permitted under points #3, for 30 minutes after the last permitted packet.

Rule 1: Allows concentration of all VK* data even when an alternate IGATE is used.

Rule 2: Allows any station that does not have a VK callsign to be treated as if they did have a VK callsign.

Rule 3: Any messages to VK stations, or other stations operating inside VK should be allowed in.

Rule 4: Allows positions of any stations sending messages to VK to show up on the server. This rule may need to be changed later so that their messages to non-VK stations do not appear – As only one side of the conversation would be heard anyway. The 30 minute limit allows keeps the link open as long as the users continue to converse.

Child to Parent IGATE

Some IGATE data is not relevant apart from for users of that IGATE. Therefore the following rules are used to filter the data stream. All packets will be passed, except if they match any of these rules.

1. Messages to 'javaMSG' and to 'USERLIST'.
2. Packets to 'FBB'
3. Packets to 'MAIL' where the first characters of the payload are 'BBS'
4. Packets that have malformed contents.

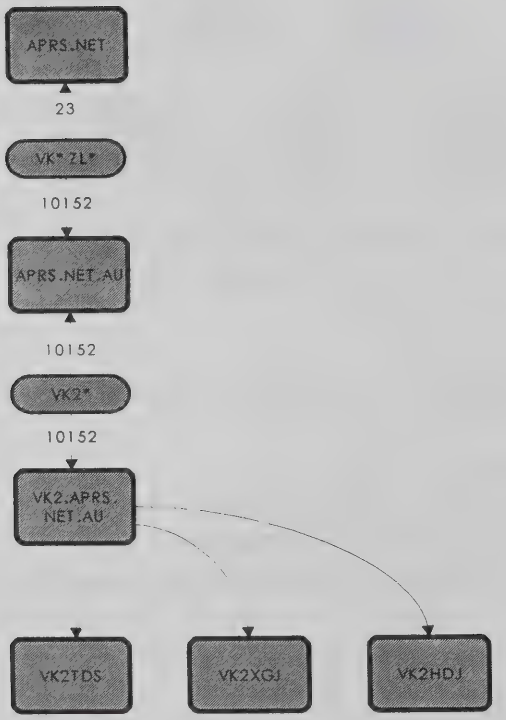
These rules reduce the amount of data being sent to the parent quite significantly.

Users?

Having a look at the diagram below one important question is 'Where does a user connect?'. The answer is basically wherever they want to. Generally the answer would be to connect to the server offering country-wide data. However if they were wanting to IGATE their information from other IGATE's in their state only, they might want to connect to their State APRS server.

In either case, any packets they receive from RF and place back onto the APRS network would filter back to the root APRS server, as well as down to the State based APRS server.

Messages will be delivered regardless of the where the user is connected, provided the connection point is within the filtering range. Messages will be fed up the chain no matter what filtering rules are in place.



Filtering Software

The filtering software I have written has been coded in Perl running under Linux. I chose Perl as it is a very quick prototyping language, with the power of 'C' and the flexibility of Basic. As it is basically an interpreted language, it is very quick to modify code and see how the changes work.

The software consists of three modules. They are

- perlFilter.pl
- perlTelnet.pl
- perlAPRS.pl

perlFilter is the main filtering program. It contains all the rules, and logic deciding what packets are allowed in either direction. It connects to the up-stream and down-stream APRSd servers, providing all the data communications between the two. perlFilter actually authenticates itself with each server so that the connections are bi-directional.

perlTelnet is a specialized TELNET program. Whilst most telnet programs quit when the connection is closed, perlTelnet then attempts a connection to the next server on the list. If no servers can be contacted, it continues until it finds one that can..

A basic structure of the software appears in the diagram following. In this diagram, the boxes with the partial callsigns ("VK*", "ZL*" and "VK2*") are the perlFilter programs, with the filter properties listed.

In this way, APRS.NET.AU only receives packets from APRS.NET that are relevant to it – All VK and ZL packets, as well as packets to those stations.

Likewise VK2.APRS.NET.AU only receives packets from APRS.NET.AU that are relevant to VK2. This arrangement reduces the load on servers and the underlying communications network.

I have also modified Richard Parry's perlAPRS software to check if the location of a mobile station is within the location specified by the configuration file.

perlFilter does NOT filter packets from stations that connect directly into the IGATE. What would be ideal is for the capability for APRSd to use external filters on all incoming data streams automatically.

Operation

The software has now been operational for several weeks, and is working well. At times it causes the operation of APRS to become less intuitive – by requiring users to send messages to stations outside the filter range before they will appear on the screen. Until the user appears on the screen, users do not normally send messages.

In other words, users must register their locations, or an association with the location (by sending a message to that location) before their data will be passed. This geo-referencing should be an integral part of all wide area AVL systems

I found that soon after I started messaging a user in the USA, his position appeared on my map. Also his messages to other users in the USA appeared on the raw data coming from my APRS server.

I found that the reduction in data volume was a great advantage in operating this way. I have found that users are generally happier with the reduced volumes, as they were being overloaded with information.

I am now looking at how to integrate similar software into APRSd as a hookpoint to improve the filtering.

Conclusion

Just as the telegraph, telephone and Internet have grown from flat networks to hierarchical networks, APRS networking must also mature in a similar manner. Unlike those other networks, the basic protocols will not need to change for this shift, just the servers, and their interconnections.

The work I have done on creating hierarchical servers is just another step in the evolution of APRS towards a scalable network oriented system. The software I have developed allows users to add customized filters to the data stream in line with local requirements.

Software Availability

The software is available on the following WWW site:
http://radio-active.net.au

It is written for Redhat LINUX, but should run on almost any platform that supports Perl and multi-processing. It does not even need to be running on the same machine as the servers it connects.

APRS®
and the
TAPR EasyTrak™
Az/El Rotor Control System

Keith Sproul, WU2Z
698 Magnolia Road
North Brunswick, NJ 08902
wu2z@amsat.org

Abstract

APRS, Automatic Position Reporting System, has been used to track many things. It has been used with DX Cluster for showing where the DX stations are located. It has been used to track cars, boats, bicycles, motorcycles, weather balloons, and hot air balloons. It has even been flown on the Energizer Bunny® Hot Air Balloon many times.

When tracking balloons, especially the high-altitude weather balloons, normal antennas work fine for the Packet/APRS/GPS signals. But when you are trying to receive ATV (Amateur Television) signals from the balloon, you need much better antennas. Usually, you need to have a high-gain antenna of some type pointed directly at the balloon.

APRS already knows where you are, and it knows where the other stations are. From this information, it is easy to calculate the angle(s) needed to point a directional antenna at the other station, when needed.

Uses of Rotor Control within APRS

Usually when people think of Rotor Control, they only think of either satellite tracking, or rotor control for DX Chasing. Although these are the primary uses, there are other situations where rotor control could be useful. The following table shows some possible uses of rotor control within the APRS system.

Satellite Tracking	Azimuth / Elevation
Balloon Tracking	Azimuth / Elevation
DX / HF operation	Azimuth only
Ground based vehicle tracking	Azimuth only

Background

APRS has been used for tracking high-altitude balloon projects for many years. MacAPRS/WinAPRS has features for predicting where the balloon will land even before the balloon is launched. This software also has display features for showing altitude. [i]

For normal APRS/GPS tracking operation with balloons, standard, omni-directional antennas do fine. The data is 1200 baud packet, and 1 – 5 watts on 2meters or 440Mhz is adequate. However, when you try to transmit Amateur Television, (ATV) signals, it is much harder to receive at great distances. Since the antenna on the balloon cannot be directional, you have to make up for it with good antennas on the ground. In the past, some groups have simply had a person manually pointing the antenna at the balloon. As long as the balloon is in view, this works fine. After that, it is just guessing. Also, this gets tiring very fast.

Computer controllable Rotor Controllers used to be complex and require special interfaces. They were also quite expensive. With the introduction of the TAPR **EasyTrak Rotor Controller**, [ii] this difficulty has been greatly reduced. This unit is also significantly less expensive than some of the other rotor controllers on the market. See the paper elsewhere in these proceedings about the details of this controller.

Software Design

If the world was flat, the calculations for tracking an object would be simple trigonometry. But since we have to deal with a round world, the math becomes more involved. In addition to having to do three-dimensional trig, we have to take into consideration the fact that the altitude reported by the GPS is the altitude above sea level at the current location of the balloon. The curvature of the earth has to be taken into consideration and subtracted out, otherwise, the antenna would be pointed too high.

The rotor itself has to be considered. Most rotors are north-centered. Depending on what you are doing, you may want a north-centered or south-centered rotor. When tracking balloons from a temporary location, you need to consider where you locate your ground station. If the rotor has to go through a 'FLIP', moving from one end of the rotation limit all the way around to the other, you will loose valuable time. During that time, the signal will be totally unusable because the antenna won't be pointing in the direction you need it to point.

Another thing to be concerned about is the fact that you don't want to constantly move the rotor, this could shorten the life of the rotor considerably. Therefore, you want to know the beam width of the antenna array, and move the antenna only when needed. Some rotor controllers already take this into account. The EasyTrack rotor controller has a setting to tell it the 'DEAD ZONE'. This is the number of degrees that the antenna has to be told to move before it will actually rotate the antenna.

Not all devices that you will want to track will be configured to send altitude. In this situation, the software realizes that the tracked device does not have altitude and does not tell the rotor controller to set the elevation angle. This allows the operator to manually adjust the elevation with the switches on the rotor controller, or manually from the software on the computer. The software will not try to point the antenna below the horizon.

The TAPR EasyTrak rotor controller uses a protocol called EasyComm. This is a simple ASCII text protocol for controlling all of the features of the rotor. This protocol is easier to use than the others and has more flexibility. It has also allowed the software to have a few more features.

Rotor Control and Status Window

WinAPRS/MacAPRS/X-APRS has a Rotor Control Window that allows you manually control the rotor and to display the current status of the rotor. This window can also be used for making sure the antenna system is working properly. The following commands are supported:

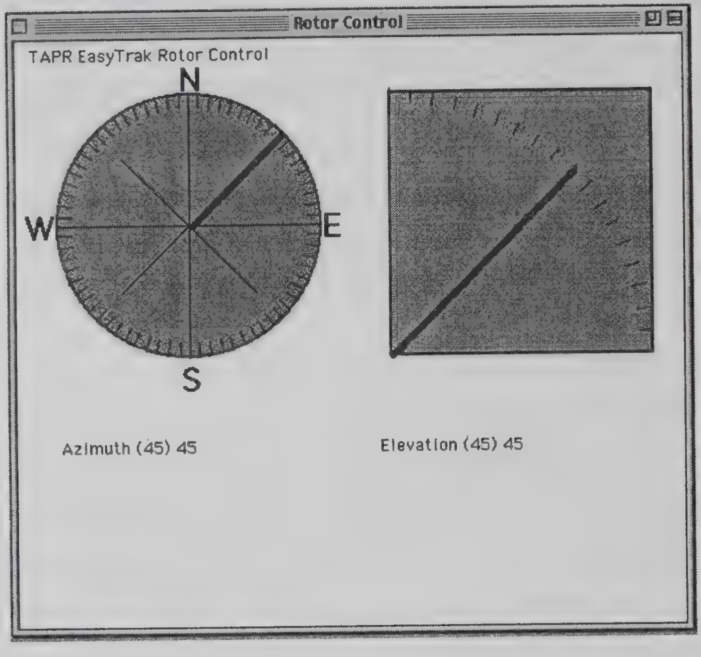
N	Set the Azimuth rotor to point North.
S	Set the Azimuth rotor to point South.
E	Set the Azimuth rotor to point East.
W	Set the Azimuth rotor to point West.
L	Rotate Azimuth rotor Left 5 degrees.
R	Rotate Azimuth rotor Right 5 degrees.
V	Set the Elevation rotor to point straight Vertical.
H	Set the Elevation rotor to point at the Horizon.
U or +	Raise Elevation Up 5 degrees.
D or -	Lower Elevation Down 5 degrees.
X or ^C	Sends a CANCEL command to the rotor.
? or /	Query the rotor controller and update the display.

The arrow keys do the same things as the ‘L’, ‘R’, ‘U’, and ‘D’ keys.

Numeric Keypad	The numeric keypad works as a control also.
5	Home (Azimuth 0, Elevation 0)
1-4, 6-9	Move the antenna in the appropriate direction, 1 degree at a time. In the direction as they appear on the numeric keypad.

There are two different configurations for the Rotor Control Window. The default window shows separate displays for azimuth and elevation. This is the used for normal manual control and display, this is also used when used for DX Cluster type applications. The other configuration shows an icon and elevation indicator. Below this display is a compass display showing which way the antenna is currently pointing. The above commands work in both configurations.

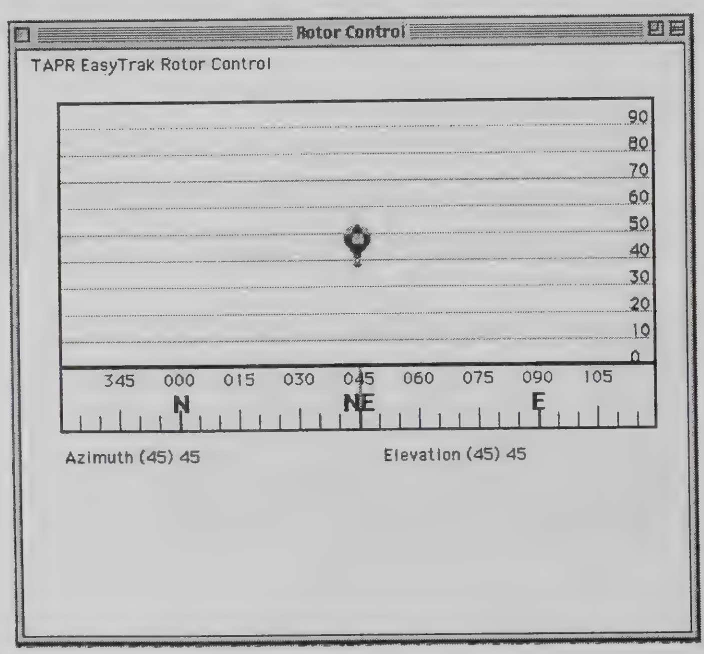
Rotor Control Window 1 (Normal Mode)



Azimuth (45) 45

Elevation (45) 45

Rotor Control Window 2 (Balloon Track Mode)



Azimuth (45) 45

Elevation (45) 45

DX Cluster Station Antenna Pointing

APRS has had the ability to understand DX Cluster data for a long time. Some people have used this feature, but it has not been a major aspect of APRS. Once rotor control ability was added to WinAPRS/MacAPRS/X-APRS, the ability to point to any station was extremely simple to add.

You can also go to any of the Station List windows, select a station there, and hit 'P', it will then POINT the rotor to that station. If the station you select does not have a valid position report, it will ignore the command. If the station has a position and altitude, it will rotate both the azimuth and elevation rotors. If the station does not have an altitude, it will simply rotate the azimuth rotor.

Note: You can make it point to any station or object in APRS, it does not have to be a DX station.

Automatic Weather Balloon Tracking

To use the Balloon Tracking feature of WinAPRS/MacAPRS/X-APRS, you need to perform the following steps:

(1) Select the station you want to track by opening up any of the Station Lists, then hi-light the station you want to track. (The TRACKED STATIONS LIST is recommended for this).

(2) Type the letter 'B' for BALLOON TRACK.

(3) Watch the rotor follow the 'balloon'

Note: The station you want to track does not have to be a balloon.

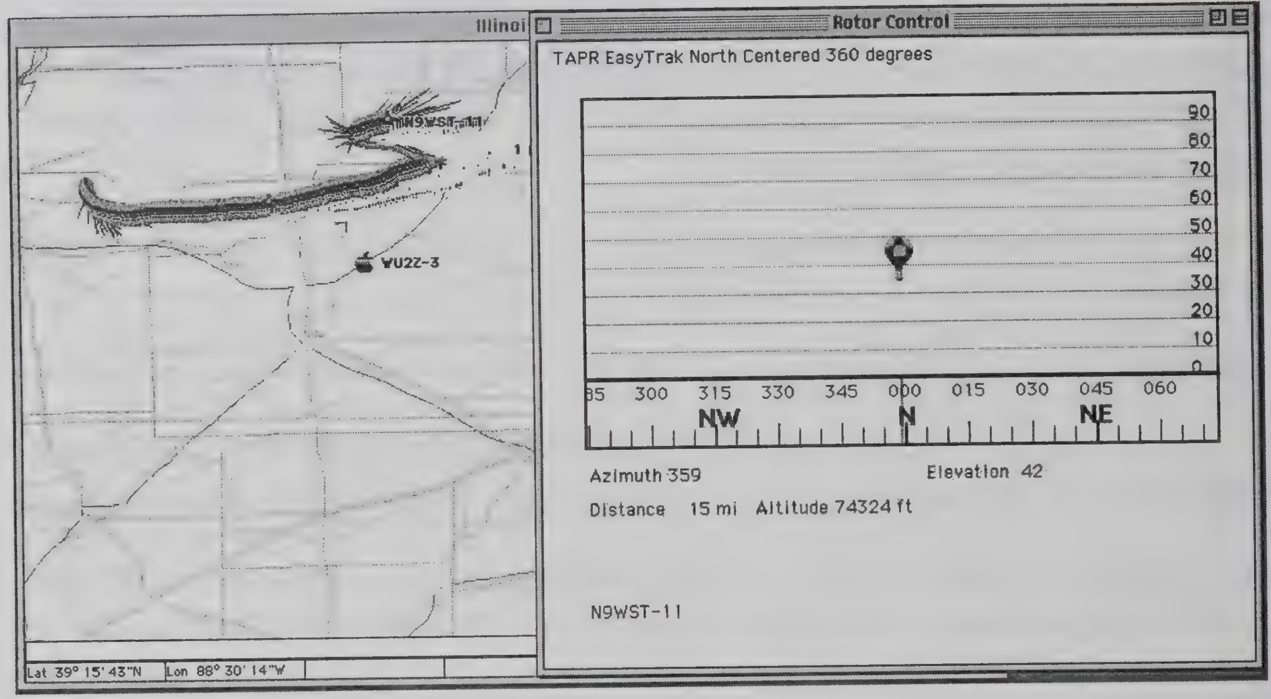
If the station is sending altitude, it will track azimuth and elevation. If the station is not sending altitude, the software will only track azimuth. If this is the case, you can manually control the elevation using the rotor control window described above.

Each time a valid position report or elevation report is received, the program will calculate the azimuth and elevation angles needed to keep the antenna pointed at the balloon, it will then send commands to the rotor controller to tell it where to point the antenna.

Automatic Weather Balloon Tracking Example

The map and rotor control windows below show a balloon in central Illinois. The track on the map shows the balloon track soon after its maximum altitude. The antenna is located at WU2Z-3 on the map. The rotor control window shows the azimuth and elevation angles that the antenna was pointed at the time the last packet was received.

Balloon Tracking Map



Lat 39° 15' 43"N Lon 88° 30' 14"W

TAPR EasyTrak North Centered 360 degrees

Azimuth 359 Elevation 42

Distance 15 mi Altitude 74324 ft

N9WST-11

Ground Based Station Tracking

In most cases, if you are tracking a moving station, it is either close enough to be heard direct, or there are digipeaters so that you are able to receive the data even when the station is far away. If you are in an area that has few digipeaters, but want to be able to track a car or other moving station, for further distances, it would be nice if you could have a directional antenna automatically point at the moving station.

To make this work, for normal APRS type operation, you would need a vertically polarized 2 meter yaggi on a rotor. Then, you select the station that you want to track, just like you would if it were a balloon. WinAPRS/MacAPRS/X-APRS will then point the antenna at the last known position of that station. Since the yagi has more gain than the normal vertical antenna, you should be able to copy the signal at much greater distances than you could if you were just using an omni-directional antenna.

Unfortunately, this cannot counter the affects of terrain. For example, this cannot compensate for a car that disappears on the other side of a hill. This feature would be most useful in areas that are flat, or when tracking boats. Since there are very few digipeaters in the water, the tracking of boats is where this could be of greatest use.

If it were desirable to track many boats, or other moving stations, at one time using this mechanism, the capabilities could be easily expanded. It would be easy to develop the software where the system would point the antenna at one station, leave it there for a couple of minutes, or until it received a position report, then rotate the antenna to the next station, wait for a position report from that station, etc. The order in which it looked at the stations would be optimized based on the least amount of antenna movement and which station had the oldest position report. The logic and code for this feature has been developed, but it is not currently in the released versions because it is doubtful if anyone would truly use it. This feature has much more probability for use in the commercial market. If anyone really wants this feature in the Ham version of WinAPRS/MacAPRS/X-APRS, please contact the authors.

Ground Based Station Tracking Example

To show how Ground Based tracking works, we conducted an experiment. We set up two stationary receive stations and one mobile station. The mobile station was configured to not use any digipeaters, i.e. the path was not set.

Receive Station 1: Omni directional antenna, about 30 ' elevation.

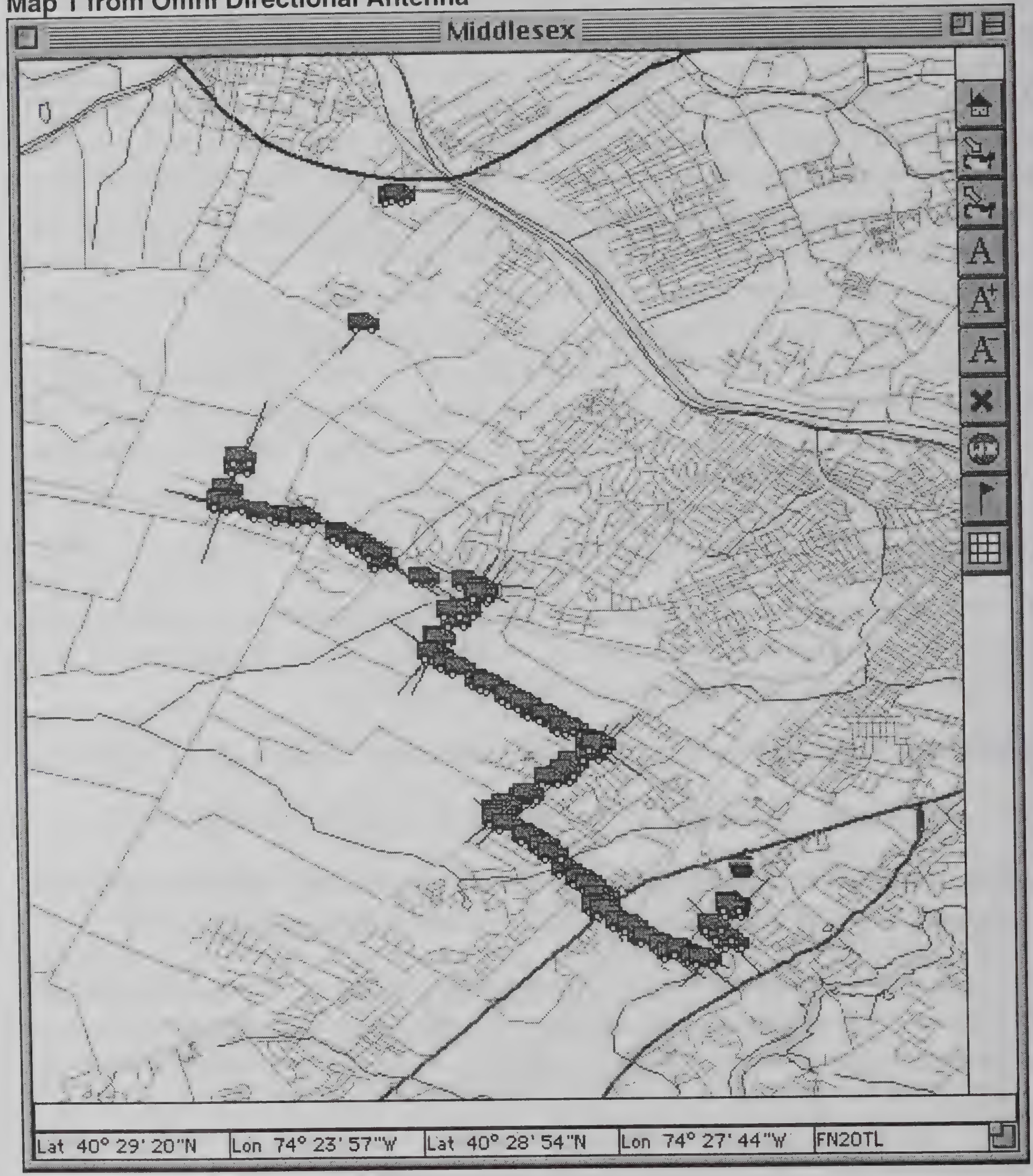
Receive Station 2: 5 element yagi, with rotor control, about 30' elevation..

Mobile Station: Omni directional antenna on my van with a Kenwood TM-D700 configured to transmit every 12 seconds, with medium power (25 watts).

This experiment was not done on 144.390 to avoid collisions with other APRS packets. This also eliminated any possibility of digipeaters. The receive site is at the Apple Icon at the bottom right of the map. The track is from my house to a hotel about 7 miles away and back.

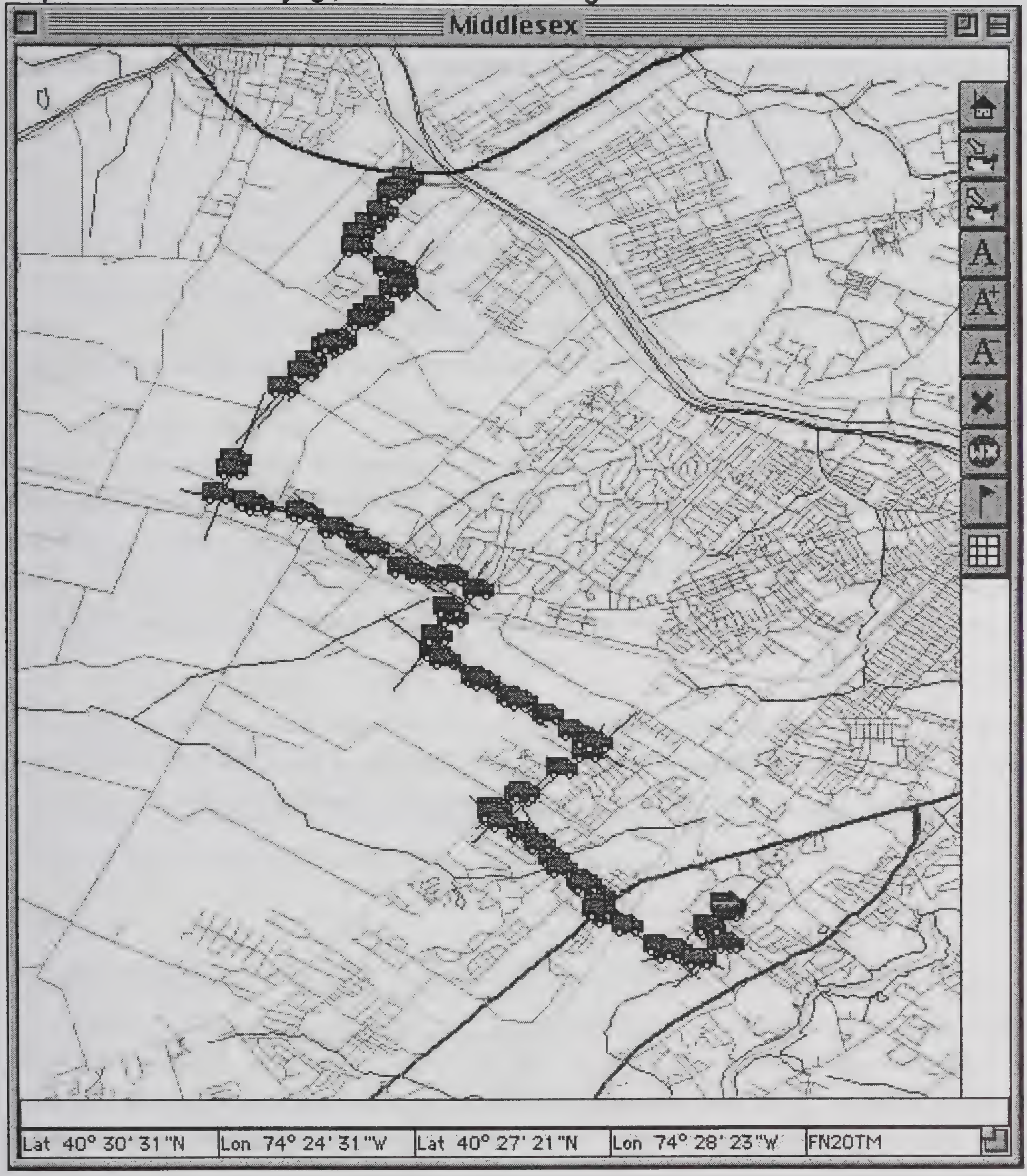
Below are the two maps showing the difference in received signals from the mobile station. Map 1 shows the position reports received on the Omni directional antenna and Map 2 shows the position reports received on the Yagi with station tracking.

Map 1 from Omni Directional Antenna



Lat 40° 29' 20"N | Lon 74° 23' 57"W | Lat 40° 28' 54"N | Lon 74° 27' 44"W | FN20TL

Map 2 from 5 element yagi, with station tracking



Lat 40° 30' 31"N | Lon 74° 24' 31"W | Lat 40° 27' 21"N | Lon 74° 28' 23"W | FN20TM

Camera Tracking

Recent events have made using this system accurate enough to keep a camera pointed at a moving object. Selective Availability has been turned off. This gives GPS an accuracy of about 30 feet. The altitude has also greatly improved.

If you have a GPS and a D7 radio and have it set to transmit your position at least once a minute or more, WinAPRS/MacAPRS/X-APRS could keep a camera pointed at you as you walk around in the area within the view camera. To make this work, the camera needs to be up high, and you have to have a fairly wide viewing angle since GPS still is not exact. When doing this, you need to get an extremely accurate position and altitude set for the camera location.

For this application, the accuracy of the Mic Encoder format is not good enough if you want to have the camera track you real close to the camera. The Mic Encoder format transmits accuracy of 1/100 of a minute. If you use a GPS data transmitter, i.e. a TNC set up to transmit the $GPGGA NMEA data, you can get much better accuracy. The Garmin III+ sends data to 1/1000 of a minute, or 10 times better than the Mic Encoder format.

The good news is that the Kenwood D7 will do this also, but you have to configure it from a computer and it doesn't remember these settings if you turn the radio off or change the TNC mode. The Palm Pilot works very good for configuring the D7 while out walking around.

The settings in the D7 needed to do this are:

GPSTEXT $GPGGA	GPGGA includes lat, lon and altitude all in one line
LOC E 3	Send the location every 30 seconds

The accuracy transmitted will now be the same as the accuracy that your GPS reports. All GPS units transmit at least 2 digits, the Garmin GPS III+ transmits 3 and I have seen a couple GPS units that actually transmit 4 digits of accuracy.

Comments

In all of the different uses of this system, it is extremely important to have the rotor system calibrated accurately, and to make sure that it is pointed at TRUE NORTH, and not Magnetic North. In New Jersey, the difference between true north and magnetic north is about 12 degrees. This can make a big difference in the accuracy of the system.

The original implementation uses the TAPR EasyTrak Rotor Controller, but the capability is being expanded to include other rotor controllers as needed.

Currently, we support:

TAPR EasyTrak

Endeaver AutoTrack

Heathkit Intellirotor Azimuth rotor only

We are working on supporting the following:

EA4TX Automatic Rotor System (ARS), From Europe (Windows Only)

Yaesu GS-232

Hygain/MFJ DCU

Kachina 505AR Azimuth rotor only

The Heathkit Intellirotor works fine for DX type operations, but due to the nature of how it operates, it is not well suited to track moving objects in real time.

Future

We are working on a rotatable antenna system on a car. The software could use the GPS data to know where you are and which way your car is headed. It would then adjust the antenna accordingly while driving. This would be most useful for chasing balloons. This is being worked on, but we need to get a useable antenna system designed before testing can start.

Garmin's eTrax Summit GPS is a GPS with pressure based altitude sensor and a flux gate compass built-in. Garmin just introduced an upgrade to this unit that includes compass headings in the NMEA data stream. This gives us an easy way to get HEADING information instead of BEARING information. This is important because the

BEARING is only accurate if you are moving. This makes the tracking from a car much easier, especially if you want to go to someplace and park, then do your tracking while stopped in the car.

For things like the camera tracker and tracking while moving in a car, it would be nice to have a much faster rotor, but currently, there are none available commercially.

Now that WinAPRS/MacAPRS/X-APRS can point an antenna, satellite tracking would be a next logical step.

Conclusion

This new addition to WinAPRS/MacAPRS/X-APRS has a lot of potential, for both the balloon chaser, and the DX person. In addition, the simple tracking of a ground base station can greatly improve the communications with stations at greater distances, especially in areas with little or no digipeaters.

As APRS continues to develop, people will find more and more interesting things to do with it.

Web Sites of Interest

http://aprs.rutgers.edu/

http://www.tapr.org/

http://www.kenwoodusa.com/

http://www.yaesu.com/

http://www.garmin.com/

http://dorm.rutgers.edu/balloons/

http://dorm.rutgers.edu/~ksproul/pantilt.html

ftp://ftp.amsat.org/amsat/software/win32/wisp/easycomm.txt

[i] ***Graphical Information Systems and Ham Radio,*** Keith Sproul, WU2Z, ARRL 14th Digital Communications Conference, Arlington, Texas, September 8-10th, 1995, pp 108-118.

[ii] ***EasyTrak, A PIC Based Rotor Control Inerface,*** Steven Bible, N7HPR, ARRL 19th Digital Communications Conference, Orlando, Florida, September 22-24th, 2000.

APRS®
and
Kenwood TM-D700 Voice Messaging

Keith Sproul, WU2Z
698 Magnolia Road
North Brunswick, NJ 08902
wu2z@amsat.org

Mark Sproul
1368 Noah Road
Nlorth Brunswick, NJ 08902
kb2ici@amsat.org

Abstract

MacAPRS has had speech capabilities for messaging for many years. WinAPRS has the ability to use wave files for some functions. APRSA Plus has wave file capabilities too. To put voice features on a normal computer is easy. Last year we wrote a paper about wanting a message 'speaking' device using some kind of text to speech system if we could find one. These do exist, but they are not cheap, and also are not very small. Last year, Kenwood introduced the TM-D700 which had the ability to do some of what we wanted, built right into the radio. This 'speech' feature has proved to be very useful. This paper is about the voice messaging feature of the Kenwood TM-D700

APRS Text Messaging

APRS has had messaging capability since the very beginning. This message system is designed for real-time, tactical messaging, and not long email messages. In the 1200 baud environment that most APRS activity operates, this works very well. You can send messages to an individual station, or you can send messages as BULLETINS, or as ANNOUNCEMENTS. The difference between Bulletins and Announcements is that BULLETINS are time sensitive and usually fairly important, such as weather warnings etc. Announcements are things such as meeting or hamfest announcements.

You can send and receive APRS messages to/from computers running APRS. The problem is that if you are running a stand-alone tracker, you cannot send or receive messages. With the introduction of the Kenwood D7 hand held radio with APRS two years ago, you could now send and receive messages from a portable unit without having to carry a full laptop computer and TNC/Radio with you. This was a great improvement for messaging with APRS. Then, last year, Kenwood introduced the D700 mobile radio with APRS features. This radio has all of the same features of the D7, but has a bigger display and somewhat easier text entry.

Now that we have APRS messaging built into our mobile radios, we have another problem. When we receive a message, we have to look at the radio to read it. If you are driving down the highway, this can be hard to do. Last year, we wrote a paper[i] about a text to speech project . The idea was to have a system that would SPEAK the message to you so you wouldn't have to look at the radio to know what the message was.

APRS Voice Messaging

The introduction of the Kenwood D-700 radio was a pleasant surprise. If you add the VS-3 Voice chip to the Kenwood D-700 radio, it will give you limited speech capabilities. The VS-3 chip is a voice chip that Kenwood has used for several years. This chip has the full alphabet, the numbers, and a few other words. It does not have text to speech capability, but it is very useful.

If you have the VS-3 chip in your D700 radio, it will spell the call sign of the sending station on all messages sent to you. It will do this for all messages sent to your call sign, regardless of SSID. This is especially useful when you are in your car. That way, you get allerted to messages sent to your home station even if you are mobile. In addition to this, if a message starts with a percent sign ‘%’, it will spell the entire message. For example, if I send a message to Bob, WB4APR, the message would be:

```
WU2Z>APRS::WB4APR   %ICU2
```

The D700 radio would then say:

‘W’ ‘U’ ‘2’ ‘ZED’ I SEE YOU TOO

In addition to the alphabet and numbers, it has some extra words. These words can be accessed by putting their codes in square brackets. For example:

```
WU2Z>APRS::WB4APR   :%ICU[6b]2[62]
```

Will say: “I SEE YOU ON TWO METER”

Kenwood D-700 VS-3 vocabulary

Code	Word	Language	Code	Word	Language
			40	0	Japanese
1	A	Japanese	41	1	Japanese
2	B	Japanese	42	2	Japanese
3	S	Japanese	43	3	Japanese
4	U	Japanese	44	4	Japanese
5	V	Japanese	45	5	Japanese
6	A	English	46	6	Japanese
7	B	English	47	7	Japanese
8	C	English	48	8	Japanese
9	D	English	49	9	Japanese
0A	E	English	4A	TEN (TEN means point)	Japanese
0B	F	English	4B	VFO	Japanese
0C	G	English	4C	MR	Japanese
0D	H	English	4D	PM	Japanese
0E	I	English	4E	Call	Japanese
0F	J	English	4F	Band	Japanese
10	K	English	50	blank (100msec)	
11	L	English	51	blank (200msec)	
12	M	English	52	High	English
13	N	English	53	Medium	English
14	O	English	54	Low	English
15	P	English	55	EL	English
16	Q	English	56	Error	English
17	R	English	57	Enter	English
18	S	English	58	Clear	English
19	T	English	59	Reset	English

1A	U	English
1B	V	English
1C	W	English
1D	X	English
1E	Y	English
1F	Z	English
20	Blank (20msec)	
21	Blank (40msec)	
22	High	Japanese
23	Middle	Japanese
24	Low	Japanese
25	EL	Japanese
26	Error	Japanese
27	Enter	Japanese
28	Clear	Japanese
29	Reset	Japanese
2A	Memory	Japanese
2B	Squelch	Japanese
2C	Power	Japanese
2D	Sharp	Japanese
2E	Star	Japanese
2F	Hello	Japanese
30	MHz	Japanese
31	KHz	Japanese
32	meter	Japanese
33	Centimeter	Japanese
34	dB	Japanese
35	Minus	Japanese
36	Plus	Japanese
37	AKI (AKI means vacant)	Japanese
38	Channel	Japanese
39	Over	Japanese
3A	Menu	Japanese
3B	On	Japanese
3C	Off	Japanese
3D	Up	Japanese
3E	Down	Japanese
3F	PF	Japanese
5A	Memory	English
5B	Squelch	English
5C	Power	English
5D	Pound	English
5E	Star	English
5F	Hello	English
60	MHz	English
61	kHz	English
62	Meter	English
63	Centimeter	English
64	dB	English
65	Minus	English
66	Plus	English
67	Open	English
68	Channel	English
69	Over	English
6A	Menu	English
6B	On	English
6C	Off	English
6D	Up	English
6E	Down	English
6F	PF	English
70	0	English
71	1	English
72	2	English
73	3	English
74	4	English
75	5	English
76	6	English
77	7	English
78	8	English
79	9	English
7A	Point	English
7B	VFO	English
7C	MR	English
7D	PM	English
7E	CALL	English

Conclusion

Kenwood continues to be an innovator in the design of radios for Amateur Radio operations. This speech capability is an added capability that has proven to be useful for the mobile APRS operator. I continue to look forward to new and innovative ideas.

MacAPRS and WinAPRS are being updated so that they will ‘speak’ these messages just like the D700 does. This way you can hear them ‘correctly’ instead of having to understand the codes and intent of the sender.

Web Sites of Interest

http://aprs.rutgers.edu/

http://www.tapr.org/

http://www.kenwood.net/

http://aprs.rutgers.edu/D700Voice.html

[i] Voice Messageing System, ARRL DCC, Septermber 1999, Phenoix Arizona, pp 100-101

Internet wide Callsign database using LDAP

Mark Sproul
1368 Noah Road
North Brunswick, NJ 08902
KB2ICI@AMSAT.ORG

Introduction:

LDAP, **L**ightweight **D**irectory **A**ccess **P**rotcol, is an internet protocol for storing directories. It has many uses but the most common use is for keeping track of large lists of personel. Many attributes can be associated with each entry such as name, email address and callsign. LDAP is a standard and is easy to incorporate support for it in any application. There are libraries for use with C/C++ Java, PHP and many other languages as well as all platforms. LDAP is also supported already in many end-user applications such as Netscape.

There are many commercial products for callsign lookup and there is the FCC database that is updated about once a week that can be downloaded (at least 70 megabytes). There are also many web sites that allow a user to do a quick call sign lookup.

However, there is no easy way for a developer to add call sign support to other applications without writing the callsign lookup code. The web based systems are intended for a user to manually type in a callsign and press a button.

By implementing a universal callsign database in an internet wide standard directory, all internet users will immediately have instant access to callsign lookup and email directory for all other hams in the US.

Background:

LDAP is Lightweight Directory Access Protcol and is a directory service. This differs from a database in that the overall design is optimized for a very high read to write ratio. A normal database has to have more equal performance for read and write. Additionally

LDAP is optimized for holding lots of small bits of information instead of large data records. LDAP was developed by the University of Michigan in response to the heavyweight X-500 system.

To get a feel for LDAP, open up the NETSCAPE Address Book and click on NetCenter, then type "sproul" into the Names Containing field, you will get a long list of people from around the country.

Just as a side note, LDAP has replaced the registry in Windows 2000 for internal storage of parameters.

Project:

The LDAP-HAM project consists of an LDAP server running on a Sun Sparc, this database will be populated and updated from the FCC database. Additional sources of information may also be used. Once everything is in place, automatic updating from the FCC database is planned on at least a weekly basis. LDAP supports database replication for optimizing performance. For example, if you have 2 large campuses, both needing high access to the LDAP server you may want to have a second LDAP server at the second campus because of limited bandwidth or because of intermittent down time. LDAP fully supports a non-real time replication system that allows one server to be the master and others to be replications.

My initial use for the LDAP-HAM server will be in augmenting email capabilities of APRS, specifically the APRS to email gateway (wu2z.rutgers.edu). Previously when the email gateway sent a message from APRS, the return address was always the senders callsign "@unknown.net". For example, if I send a message to someone, the FROM address would show up as kb2ici@unknown.net. This is because there is no knowledge of a persons email address within APRS. By having the LDAP server, it will be real easy for the email gateway to execute a quick lookup and get the actual email address for KB2ICI and then substitute that address (which is kb2ici@amsat.org) in place of the kb2ici@unknown.net.

Server:

The LDAP-HAM server is running slapd from the OpenLDAP Foundation (http://www.openldap.org/). OpenLDAP is full open source and runs on many platforms. There are other LDAP servers available commercially from Netscape and from Microsoft.

Web access:

There will need to be a simple access to the data for casual lookups and there will need to be one or more methods for updating changing information, primarily email address. This is implemented with an apache web server and php (http://www.php.net).

Access by others:

One of the key goals of this project is for other developers, both traditional application developers and web developers, to be able to have simple and free access to the server. Modification of the data on the server will of course be password protected but read access will be open to anyone.

Sources:

The LDAP server, slapd, is open-source and is available from the OpenLDAP Foundation. The same goes for the replication server, slurpd

The web interfaces are written in PHP, most or all of these sources will be made available, contact the author.

References:

LDAP	**L**ightweight **D**irectory **A**ccess **P**rotcol http://www.umich.edu/~dirsvcs/ldap/
OpenLDAP	OpenLDAP Foundation http://www.openldap.org
PHP	Hypertext Preprocessor http://www.php.net/

<APRSdec>™ — The G3NRW APRS Packet Decoder

by Ian Wade, G3NRW (email: g3nrw@tapr.org)

7 Daubeney Close, Harlington, Dunstable, Bedfordshire LU5 6NF, United Kingdom.

4 July 2000

Abstract

This paper describes the principal features of the <APRSdec> program. The program fully decodes raw APRS packets, producing reports in plain English. It has proved to be an extremely useful APRS diagnostic and learning tool. <APRSdec> is written in Perl, and runs under native DOS, Windows, Unix and Linux. It is available from `http://www.tapr.org/~g3nrw`.

Introduction

<APRSdec> started life towards the end of 1999, when I was working on the APRS Protocol Specification. I needed a simple tool to decode the many different types of APRS packet seen on air, to help me verify that I understood all the formats. Later, in the spring and early summer of 2000, the program unexpectedly came into its own as a diagnostic tool, when floods of badly corrupted packets appeared on the APRS network (arising from a combination of broken APRS client software and broken IGate server software). <APRSdec> was invaluable in reporting the errors and providing clues towards identifying the culprits.

<APRSdec> is a particularly useful tool for:

- APRS software development and testing.
- PIC development and testing.
- Network fault-finding and diagnostics.
- Learning about the APRS protocol.

<APRSdec> Features

- Runs under native DOS, Windows 95/98 and Linux/Unix. (It will almost certainly run under Windows NT and Windows 2000 as well, but this has not been tested).
- Accepts raw input in TNC/IGate terminal output format.
- Accepts raw input in UI-View two-line log format, and decodes the 15-digit UI-View timestamp.
- Performs rigorous format checking, with detailed error reporting.
- Understands position ambiguity, reporting the bounding box in which the station is located.
- Compares lat/long position against a prefix/country database — if the station's position appears to be outside the country, <APRSdec> reports a possible anomaly.
- Reports data values in imperial, nautical and metric units.
- Fully decodes Mic-E and compressed position formats.
- Recognizes data from Kenwood TH-D7 and DM-700 radios, and changes the incorrect DM-700 APRS Data Type Identifier to "Current Mic-E Data".
- Provides detailed weather station reports, including the calculation of windchill and dew point.
- Decodes storm data.
- Decodes bearing and range data.
- Decodes DX Cluster reports, showing the data as it will appear on TH-D7 and DM-700 screens.

Some Examples of <APRSdec> Output

A simple lat/long report:

```
-------------------------------------------------------------
Record #170
KE4SZT>APRS,W4NHV-9,WIDE*,WIDE/2:@270700z3408.76N/07924.60W-  John in Marion, SC-793-
 APRS Data Type= Posit w/ time. With APRS
 Day= 27  Time= 07 hours 00 mins UTC
 Lat= 34 deg 08.76 min N  Long= 79 deg 24.60 min W
 Icon= House QTH VHF   Overlay= (none)
-------------------------------------------------------------
```

A Mic-E report. <APRSdec> recognizes this comes from a Kenwood TM-D700 radio and automatically corrects the APRS Data Type to "Current Mic-E data":

```
-------------------------------------------------------------
Record #8051
KC7EQL>VE7VAN-3*>WIDE3-1>T8PWPW:'37]lIe-/]"55}Mark at Home awaits Sunsat
 APRS Data Type= Current Mic-E data
 Radio= Kenwood TM-D700
 Message Type= /M2/In Service
 Lat= 48 deg 07.07 min N  Long= 123 deg 27.65 min W
 Icon= House QTH VHF   Overlay= (none)
 Course= 173 deg   Speed= 4 knots (4.6 mph  7.4 kph  2.1 m/s)
 Altitude= 397 feet (121 meters)
-------------------------------------------------------------
```

A compressed lat/long position, within an APRS Object report:

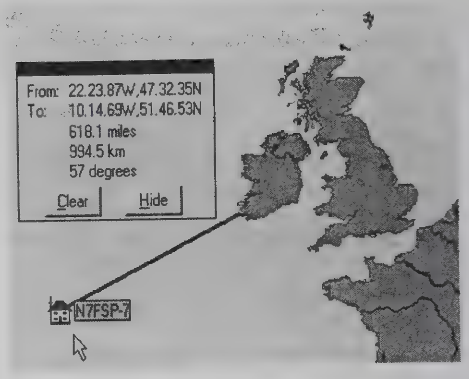
```
-------------------------------------------------------------
Record #1763
N9IDH>APRK11,AE9A-10*,WIDE3-2:;AO16     _111826z\%H+!JvA!Sk{!R=05280/437.051/13Khz/APRStk
 APRS Data Type= Object
 Object Name= 'AO16     '  Object Status= Killed
 Day= 11  Time= 18 hours 26 mins UTC
 Lat= 81 deg 14.20 min N  Long= 14 deg 04.30 min W
 GPS Fix= Old (last)   NMEA Source= Other   Compression Origin= Compressed
 Course= 296 deg   Speed= 1018 knots (1171.5 mph  1885.3 kph  523.7 m/s)
 Icon= Satellite/PAC   Overlay= (none)
-------------------------------------------------------------
```

A Long Way from Home

This UI-View report (complete with the decoded UI-View timestamp) shows APRS Working Group Chairman John Ackermann, N8UR, jogging towards G3NRW's home at over 72mph — he was on a train at the time! <APRSdec> recognizes that John is outside the contiguous 48 states, so it questions the reported location:

```
-------------------------------------------------------------
Record #622
N8UR-7*>RELAY>WIDE>WIDE>WIDE2-2>U1UVXX (UI):`v9f"> [/>"58}
 UI-View Timestamp= 27 May 2000   12 hrs 58 mins 32 secs (PC clocktime)
 APRS Data Type= Current Mic-E data
 Radio= Kenwood TH-D7
 Message Type= /M2/In Service
 Lat= 51 deg 56.88 min N  Long= 0 deg 29.74 min W
**QUESTION: Is this lat/long position reasonable?
            It seems a long way from home for this callsign.
 Icon= Jogger   Overlay= (none)
 Course= 1 deg   Speed= 63 knots (72.5 mph  116.7 kph  32.4 m/s)
 Altitude= 407 feet (124 meters)
-------------------------------------------------------------
```

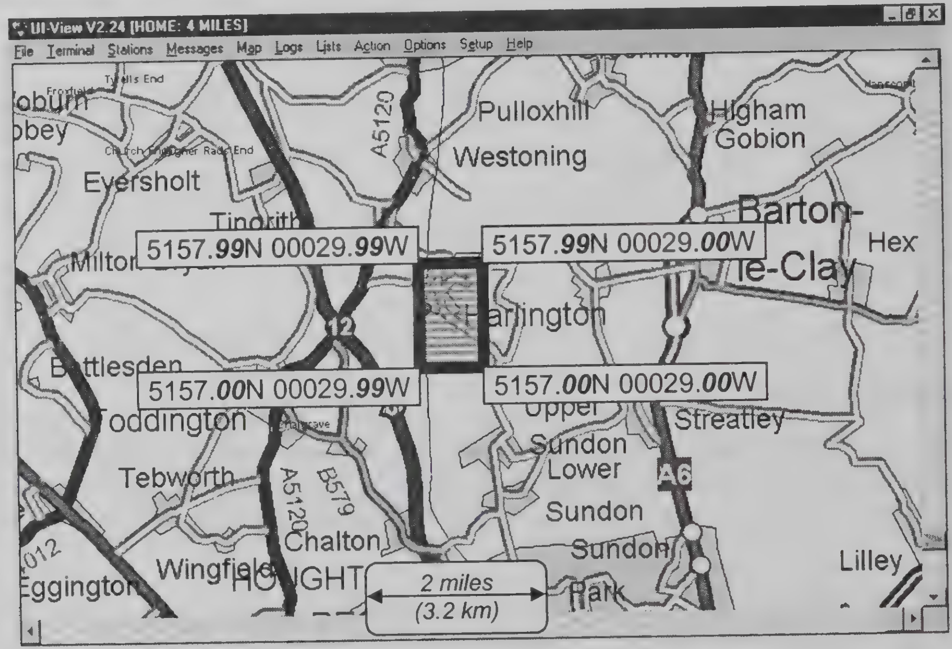
The next report is an example of how data was corrupted somewhere in the APRS network — the leading longitude digit was somehow changed from 1 to 0, making N7FSP-7 appear in the Atlantic Ocean, 600 miles off the southwest coast of Ireland:


From: 22.23.87W,47.32.35N
To: 10.14.69W,51.46.53N
618.1 miles
994.5 km
57 degrees

```
-------------------------------------------------------------
Record #646
N7FSP-7>APK101,N7OEP-10*,WIDE3-2:@111816z4732.21N/02222.65W-
                                   000/000/Mic-E/M0/Off duty>
 APRS Data Type= Posit w/ time. With APRS
 Day= 11  Time= 18 hours 16 mins UTC
 Lat= 47 deg 32.21 min N  Long= 22 deg 22.65 min W
**QUESTION: Is this lat/long position reasonable?
           It seems a long way from home for this callsign.
 Icon= House QTH VHF   Overlay= (none)
 Course= (Indeterminate) deg   Speed= 000 knots (0.0 mph
                                            0.0 kph  0.0 m/s)
-------------------------------------------------------------
```

Position Ambiguity

Position ambiguity allows a station to specify its approximate rather than absolute position. One way to do this is to replace coordinate digits with spaces. For example, G3NRW's home to the nearest minute of latitude and longitude could be specified as 5157.␣␣N and 00029.␣␣W (where ␣ represents a space). This means that the latitude may be anywhere between 51° 57.<u>00</u>′ and 51° 57.<u>99</u>′ N and the longitude may be anywhere between 000° 29.<u>00</u>′ and 000° 29.<u>99</u>′ W. In other words, G3NRW is located somewhere inside the bounding box shown below:



<APRSdec> understands this position ambiguity, and reports the coordinates of opposite corners of the box:

```
-------------------------------------------------------------
Record #1
G3NRW*>APRS:!5157.  N/00029.  W-
 APRS Data Type= Posit w/o time. No APRS
 Ambiguous position. Opposite corners of bounding box:
  NW Corner: Lat= 51 deg 57.99 min N  Long= 0 deg 29.99 min W
  SE Corner: Lat= 51 deg 57.00 min N  Long= 0 deg 29.00 min W
 Icon= House QTH VHF   Overlay= (none)
-------------------------------------------------------------
```

<APRSdec> similarly reports a bounding box for a Maidenhead grid locator:

```
-------------------------------------------------------------
Record #459
CT1DBH-1>ID:[IM58IS]
 APRS Data Type= Maidenhead Grid Square
 Maidenhead locator bounding box:
  NW Corner: Lat= 38 deg 47.50 min N  Long= 9 deg 20 min W
  SE Corner: Lat= 38 deg 45.00 min N  Long= 9 deg 15 min W
**NOTE: This Maidenhead format with square brackets is obsolete.
-------------------------------------------------------------
```

Here is a full WX station report, with computed windchill and dewpoint:

```
-------------------------------------------------------------
Record #5034
KF0ZH*>APR819,GATE,WIDE2-2:@111955z4447.06N/09329.48W_125/008g009t079r000p003....h41b10190dU2k
 APRS Data Type= Posit w/ time. With APRS
 Day= 11  Time= 19 hours 55 mins UTC
 Lat= 44 deg 47.06 min N  Long= 93 deg 29.48 min W
 Icon= WX station   Overlay= (none)
 Wind Direction= 125 deg   Speed= 008 knots (9.2 mph  14.8 kph  4.1 m/s)
 Gust Speed= 9 mph (14.5 kph  4.0 m/s  7.8 knots)
 Temp= 79 degF (26.1 degC)   Windchill= 76.5 degF (24.7 degC)
 Rain: Last hour= 0 in (0.0 mm)   Last 24 hrs= 0.03 in (0.8 mm)
       Since midnight= (Indeterminate)
 Humidity= 41 percent   Dew Point= 53.2 degF (11.8 degC)
 Barometric Pressure= 1019 mbar/hP
-------------------------------------------------------------
```

Finally, here is an example of a DX Cluster report, showing how the data appears on Kenwood radio displays:

```
-------------------------------------------------------------
Record #3852
NA4V-3>RESORC,AB4KN-2*,WIDE3-2:DX de NA4V-3   >FO29@1509 FO20@1503                           SATS
 DX Cluster Information:

   TH-D7 Screen 1     TH-D7 Screen 2               TM-D700
 +---------------+  +---------------+    +-----------------------------+
 | Dx:FO20@1503  |  | Dx:FO20@1503  |    | 1:FO20@1503 FO29@1509   SATS|
 | FO29@1509     |  |               |    |                             |
 |          SATS |  |               |    |                             |
 +---------------+  +---------------+    |                             |
                                         +-----------------------------+

-------------------------------------------------------------
```

Input Files

<APRSdec> understands input data in regular TNC/Igate terminal format: e.g .

```
W9ZGU>APR846,WA4WHD-3*,WIDE/V:=2559.  N/08010.  W-PHG6560/Carl's Castle!
```

and in two-line UI-View log format: e.g.

```
36673.5168518519pM1DTA>MB7UPG>G1YFF*>TRACE3-1>APRS [UI]:
=5143.43N\00035.88EUM1DTA Bob on Danbury Hill. Email address: m1dta@yahoo.com
```

To obtain raw data from an IGate, you can telnet to it; e.g. telnet www.aprs.net. However, many APRS report lines are much longer than the 80-character line limit of most telnet clients. These lines will be split, so that some reports may occupy 2 (or even 3) separate lines. It is possible to join them up again using a text editor, but that is a long, tedious process!

Much better is to use a telnet client that can read long lines without wrapping or truncating. By far the best I have seen is *TeraTerm Pro*, from `http://hp.vector.co.jp/authors/VA002416/teraterm.html`.

Throughput and Probability of Correct Message Transfer of the Pactor-II System Measured on Near-Vertical-Incidence-Skywave (NVIS) Paths

Ken Wickwire (KB1JY), Mike Bernock (KB1PZ) and Dave Willard (W1EO)

1. Introduction

This paper is another in our series treating on-air measurement of throughput in characters per second (cps) for various HF data-transmission protocols of interest to amateurs (see the references to our other reports at the end of the paper). Here we describe an extensive set of measurements of throughput for compressed and uncompressed text files sent over near-vertical-incidence-skywave (NVIS) paths with the Pactor-II data transfer protocol. The implementation we used was the one in the Special Communications Systems (SCS) modem.

NVIS paths, which often experience relatively difficult channel conditions characterized by multipath interference, high noise, nighttime QRM and midday absorption by the so-called D-Layer, are used to communicate over roughly 20- to 300-mile ground distances using antennas that can launch energy at high takeoff angles (horizontal dipoles, sloping longwires, bent-over whips, etc.). Since conditions on NVIS links are almost always worse than those on longer one-hop skywave links, evaluations of performance on NVIS links provide conservative (lower bound) predictions of skywave performance.

Despite its relatively high price ($650-$950), the Pactor-II system has become popular in amateur circles as an important hardware component of the admirable Winlink and Airmail HF E-mail systems. These systems are used by hams in sailboats and RVs to keep in touch with fixed (often non-amateur) sites via a combination of HF and the Internet. Two versions of the Pactor-II modem are available, the PTC-II and the PTC-IIe. These differ mainly in the "non-Pactor-II" features (packet modem slots, rig control interface, text display on modem front panel) that come with the PTC-II but not with the PTC-IIe. All of our testing used PTC-IIs with firmware versions 2.7 or later installed (the newest firmware as of this writing is version 3.0).

Pactor-II is an improvement of the widely used Pactor-I frequency shift keying (FSK) system. Pactor-II uses a two-tone, differential phase-shift keyed (DPSK) waveform whose signal-constellation size is chosen from among two, four, eight and sixteen phase shifts (DBPSK, DQPSK, 8-DPSK and 16-DPSK). The corresponding raw (channel) data rates are 200, 400, 600 and 800 bits per second (bps).

Error control is accomplished in two ways in Pactor-II: with forward error correction (FEC) and with an automatic repeat request (ARQ) scheme. (This powerful combination is employed by almost every modern point-to-point HF data communications system.) The FEC uses an interleaver, a constraint-length-9 convolutional encoder and a Viterbi soft-decision decoder.

As the Pactor-II protocol adapts itself to changing channel conditions in ARQ (connected) operation, it chooses its modulation mode from those described above, along with its FEC coding rate, which is the ratio of data to FEC coding bits in data packets. These two kinds of change are coupled, so that as the size of the signaling constellation goes up (when the protocol determines that conditions are good), the coding rate also goes up (i.e., less coding overhead is sent with each packet than is sent in poorer conditions). This has the effect of pushing more data through the channel when it's good and slowing things down when there are many bit errors and many frame repeats are requested by a receiving station. The net result of the changes is achievement of roughly the highest protocol throughput the channel will support at any particular time.

As with most other commercial adaptive HF data communications systems, details on precisely what the Pactor-II protocol measures to adapt itself to channels are proprietary. The same is true of the algorithms that do the adaptation, since such algorithms take a great deal of thought and experimentation to perfect.

The Pactor-II ARQ scheme takes advantage of the soft (analog) decisions made by the FEC decoder to perform "memory ARQ." In this technique, a data packet corrupted by so many channel errors that the FEC scheme can't correct it is saved and subsequently combined with other repeats of the same packet to reconstruct an error-free version of the original sent packet. All packets received in the ARQ mode are checked for errors with a relatively powerful 16-bit cyclic redundancy check (CRC) sequence.

As a further throughput-enhancing device the system uses various (settable) modes of data-compression, applied on a packet-by-packet basis. There are three compression techniques. These are run-length, Huffman and "pseudo-Markov coding" (PMC). Run-length compression is used when there are long series of repeated bytes in a file; for example, underscores. Huffman compression uses language-specific character-occurrence statistics to assign short bit-codes to characters (like "e" in English texts) that appear frequently. PMC is an advanced compression technique that codes pairs of commonly occurring characters rather than single characters, as in the Huffman approach.

The compression modes are

- Mode 0: 8-bit ASCII (no compression).
- Mode 1: 7-bit ASCII with Huffman compression or 8-bit ASCII, whichever makes the message smaller.
- Mode 2: Run length, Huffman and PMC, or 8-bit ASCII when required.

Compression is generally a good idea when sending files in the "connected" (ARQ) mode. Exceptions are executable and certain already-compressed (e.g., graphics) files, which sometimes actually get bigger when compression algorithms are applied to them. Compression is not normally used in the broadcast (non-ARQ) mode since in that mode receivers have no way of asking for repeats of frames that arrive with corrupted compression information. To get a good understanding of the average throughput in each compression mode we gathered a large amount of data in each mode.

Various user interfaces are available for operating Pactor-II modems, including those that allow one to send graphics and other "binary" (eight-bit-character) file data, in addition to text files. To send files in our testing we wrote C-code that interfaced PCs and Macs directly with the modems using the modems' built-in command set. We received files in the Pactor-II and Airmail mailboxes.

The Airmail mailbox resides on PCs running the Airmail software. The Pactor-II mailbox is in the Pactor-II modem itself. We treated the two mailbox systems separately because we did not know at the outset of our testing if the two systems would produce the same performance in the same conditions. (More on this later.) For further details on how Pactor-II works, see the documentation supplied with the modems and at the voluminous SCS website (URL: http://www.scs-ptc.com).

The NVIS stations we used for our measurements are 30 to 110 miles apart, and are located in Massachusetts, New Hampshire and Maine. NVIS paths often display strong multipath, high local and propagated noise, D-layer absorption at mid-day and occasionally strong interference from other stations operating in both voice and digital modes. (Horizontal antenna polarization at all our NVIS stations allowed us to be pretty confident that we were using NVIS rather than surfacewave propagation, and this was confirmed by the fading and other skywave propagation phenomena we observed during numerous file transfers.)

We tested Pactor-II in the ham bands and also on assigned frequencies outside the ham bands. The frequencies used were between 3 and 10 MHz, with frequencies in the lower half of this range being used a night. Since the average sunspot number was high (over 100) during most of the test period, solar flares and other magnetic activity were relatively common. We were thus able to enjoy the higher frequencies that come with increased sunspot activity while also observing near or complete propagation blackouts caused by solar flares. It was an interesting time for NVIS testing. The tests covered the period from June 1999 to August 2000.

We used standard Yaesu (FT-1000MP, FT-757GX) and Icom (IC-751A) amateur transceivers, all running about 100 watts. On radios equipped for it (the 1000MPs) we used 500-Hz receiving filters. A discussion of the antennas we used is in the next section. We ran tests during both day- and nighttime using automated testing and data-logging software we wrote for that purpose.

Daytime was defined to occur between the fixed times of 1000 and 2200 GMT (5:00 AM to 5:00 PM local time). Note that because our measurements were made over the course of more than a year, "nighttime" (5:00 PM to 5:00 AM local time) was not always associated with darkness at the path midpoint. Nevertheless, most nighttime measurements were taken in conditions that characterize conventional nighttime HF propagation: high noise and increased interference.

Frequencies ten or twenty percent below the so-called Maximum Usable Frequency (MUF) are traditionally thought to be the best ones to use for HF. Since we achieved the goal of working just below the MUF only approximately with our available frequencies, our results are conservative. A properly set up automatic link establishment (ALE) system that used quasi-real-time channel assessments to choose the best operating channel for the Pactor-II waveform might have allowed us to avoid the "far-from-the-MUF" conditions we occasionally encountered and thus achieve higher throughput. We say "might" because Mil-Std-188-141A ALE and Pactor-II use different waveforms. The extent to which channel-quality measurements made with FSK ALE can predict the best frequency to use with PSK Pactor-II is not clear.

The rest of the paper describes the paths between stations and antennas, the automated test software and our file-transfer operations, the recorded data format, our special-purpose statistical analysis software, a statistical summary of the data, a discussion of the statistical results and concluding remarks.

2. Layout of Paths and Discussion of Antennas

The stations used for the NVIS tests are in Bedford, Mass. (KB1JY/BED1), Derry, N.H. (KB1PZ/DER1), and Portland, Maine (KB1JY-1/POR1). Figure 1 gives the layout of the stations. Bedford used 80m dipoles up 30 feet. Derry used an off-center-fed 80m dipole up 30 feet or a terminated sloping longwire about 180 feet long. Portland used an end-fed

120-foot unterminated longwire pointed southwest. The links (followed by lengths and rough estimates of the percentage of data collected over each link) are

NVIS Links

- Bedford-Derry (30 miles, 30%)
- Portland-Derry (60 miles, 30%)
- Portland-Bedford (110 miles, 40%)

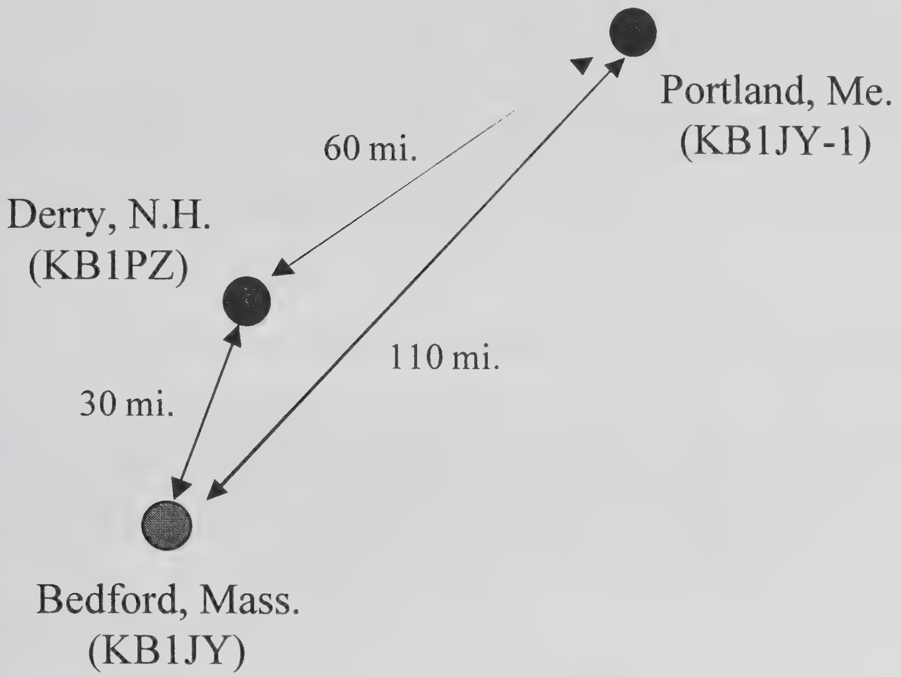


Fig. 1. Layout of Our NVIS Stations

3. The Automated Test Programs and Test Flow

To automate our testing we wrote two programs that connect to the Pactor-II and Airmail mailboxes, send the appropriate send-message command and message subject at the mailbox prompt, upload a canned text message of known size, close the link and log various statistics of the transfer to an archive and a log file. (Two programs were necessary because of minor differences in the prompts issued by each of the mailboxes.)

Since we used both PCs and Macs in our testing we actually wrote a total of four test programs. (Their modular construction makes them look very similar.) The PCs ran 16-bit executables in DOS console windows and the Macs ran standard console applications. The corresponding pairs of PC and Mac programs look almost identical. Only the routines for setting up comports and for character I/O were different since the two operating systems use different approaches to serial-port I/O (direct access to hardware and do-it-yourself interrupt handling in DOS and the Toolbox serialcom API for everything on the Mac).

The archive contains one line of performance data for each transfer. The data in these lines are discussed below. The archive also contains data-lines for transfers that timed out (failed); such lines are used to calculate transfer-success probabilities. The log file (written also to the console for monitoring of test progress) contains details of each message transmission and is normally overwritten by log data for the next transfer. We used data in the log file to fine-tune the testing software and occasionally to shed light on anomalous protocol behavior.

Shown below is a log file for a successful transfer of a 25k text file to a Pactor-II mailbox. The file contains a time-tagged narrative of what happened during the transfer and concludes with performance statistics that are similar to those that go into the performance archive file. Characters in square brackets in the log file are those sent by the local (file-sending) Pactor-II modem to the local PC running the test program. This is the character stream that is monitored by the test program for events that trip actions like sending a message-subject string or recording the message transfer time. Square-bracketed characters appearing after the first set have been suppressed to save space.

```
No xfers = 1 Time between xfers = 1 s
PTC-II TRANSFER(S) @ BED1 TO PTC-II MBOX v. macPTC26 Mode = 2
30.06.00 15:52:09 C DER1

[0xD][0xA][c][m][d][:][ ][0xD][0xA][*][*][*][ ][C][L][R][0xD][0xA][0xD][0xA][c][m]
[d][:][ ][0xD][0xA][*][*][*][-][P][A][C][T][O][R][][T][R][A][N][S][M][I][S][S][I][O][N]
[-][M][O][D][E][ ][(][0][=][A][S][C][I][I][/][1][=][H][U][F][F][M][A][N][/][2][=][P][M]
[C][)][:][ ][2][0xD][0xA][0xD][0xA][c][m][d][:][ ][0x1E][0xD][0xA][*][*][*][ ][N][O]
[W][ ][C][A][L][L][I][N][G][ ][D][E][R][1][0xD][0xA][0x1E][0x1E]
[*][*][*][C][O][N][N][E][C][T][E][D]
30.06.00 15:52:11 Connected

30.06.00 15:52:20 BBS prompt rcvd
30.06.00 15:52:20 S BED1 file.25k

30.06.00 15:52:33 Subject prompt rcvd, sending file.25k...

msg file size = 25000 uploads = 3 last_upload = 6006
30.06.00 15:52:45 0xFFA4 slave CHO
30.06.00 15:52:45 0xFFAA master traffic
30.06.00 15:52:45 0xFFA9 master request
30.06.00 15:52:45 0xFFAA master traffic
...
30.06.00 15:55:30 rem read 1 10000 chars sent, 9000 rem-echoed chars rcvd
...
30.06.00 15:55:58 0xFFA9 master request
30.06.00 15:56:02 0xFFAA master traffic
30.06.00 15:56:06 0xFFA8 master error
...
30.06.00 15:58:36 rem read 2 19000 chars sent, 18000 rem-echoed chars rcvd
30.06.00 15:58:44 Sending EOM
...
30.06.00 16:00:07 rem read 3 25000 chars sent, 25000 rem-echoed chars retrieved
...
30.06.00 16:00:10 Message "file.25k" filed.
LinkTime 2 s NegotiationTime 22 s
```

```
25006 bytes sent in 456 s; 54.8 bytes/s 35 TxARQs 1 TxERRs
30.06.00 16:00:10 BYE sent...
...
30.06.00 16:00:21 Disconnected
Trial 1: Successful xfer to PTC-II MBOX
```

The log starts by recording the number of scheduled message transfers (here one) and the time between transfers (entered here as one second but ignored since only one transfer is scheduled). This is followed by a banner announcing which mailbox system is being called, the software version and the compression mode (Mode = 0, 1 or 2; see Section 1).

To send a file using the built-in Pactor-II command set one first establishes an ARQ link with the receiving station by sending "C callsign" to the local modem, where callsign applies to the called station. The log file records this action with the line

```
30.06.00 15:52:09 C DER1
```

Once the link is established (which is usually confirmed in a second or two by a

```
***CONNECTED
```

string from the local modem), the test program answers the BBS prompt with the appropriate "Send callsign…" command. For the built-in Pactor-II mailbox this is the callsign of the message recipient followed by the title of the message. (We send to our own callsign at the remote mailbox to give ourselves deleting permission in the mailbox) This "send-sequence" is recorded in the example above with the lines

```
30.06.00 15:52:20 BBS prompt rcvd
30.06.00 15:52:20 S BED1 file.25k
```

After transmission of the send command, the test programs calculate and record the size of the message to be sent, including the end-of-message (EOM) string:

```
msg file size = 25000 uploads = 3 last_upload = 6006
```

The number of "uploads" has to do with our method of flow control. To prevent overflow of the local (sending) modem's 12k transmitting buffer, we do a first upload of at most 10k characters followed by subsequent uploads (if the file was big enough to require it) of 9k characters each. (More sophisticated flow-control is possible in the PTC-II hostmode.) After each upload, subsequent uploads to the buffer are started only after a count of 9k "echoed" characters shows that all but 1k of the previous upload has been received and acknowledged by the remote station. (Activating the desired "remote-echo" mode is done by setting the modem to "Terminal Mode 1." Note that remote-echoed characters are not sent over the air; they are simply sent by the local modem to the local PC in response to acknowledgments from the remote station.) Leaving 1k in the buffer ensures that the modem always has characters to send until the message is gone.

In the example above the message file is 25k bytes long, so the first upload is 10000 byes, the second is 9k bytes and the third is 25k - (10k + 9k) + (EOM size) bytes. Since the EOM we send to the Pactor-II mailbox is *\rnnnn\r*, where \r is a linefeed, the third and last upload is 25000 – 19000 + 6 = 6006 characters long, as indicated.

After the last upload is finished (but before all the characters in the transmit buffer have been echoed), the software sends the EOM:

```
30.06.00 15:58:44 Sending EOM
```

Shortly after the beginning of the first upload the modem starts sending the contents of its TX buffer. The test software also begins counting remote-echoed characters for flow control, and recording "status bytes" sent over the serial port by the local modem.

Status bytes reflect link changeovers (CHOs) sent by the master ("master CHO") or received from the slave ("slave CHO"). They also indicate when traffic has been sent by the master ("master traffic"), master/slave idle states, frame repeat requests from the slave (labeled "master request" since the corresponding status byte is recorded *by the master*) and erroneous frames received from the slave. The latter are labeled "master error" since the corresponding status byte is again recorded *by the master*. Request and error counts give a useful record of how hard the protocol worked to deliver messages. In the example above, the log file recorded

```
30.06.00 15:52:45 0xFFA4 slave CHO
30.06.00 15:52:45 0xFFAA master traffic
30.06.00 15:52:45 0xFFA9 master request
30.06.00 15:52:45 0xFFAA master traffic
...
30.06.00 15:55:30 rem read 1 10000 chars sent, 9000 rem-echoed chars rcvd
...
30.06.00 15:55:58 0xFFA9 master request
30.06.00 15:56:02 0xFFAA master traffic
30.06.00 15:56:06 0xFFA8 master error
...
30.06.00 15:58:36 rem read 2 19000 chars sent, 18000 rem-echoed chars rcvd
30.06.00 15:58:44 Sending EOM
...
30.06.00 16:00:07 rem read 3 25000 chars sent, 25000 rem-echoed chars retrieved
...
```

The actual status byte sent by the modem is written to the log file in hexadecimal format (e.g., 0xFFA9) followed by an explanatory label (e.g., "master request"). If repeat requests and erroneous frames from the message-receiving station are received (the former is quite common and the latter relatively rare) they are time-tagged, counted and logged as illustrated above.

Recall that the "master" label here means that the corresponding status was *logged by the master* (the local, message-sending station). A repeat request ("master request") or received error ("master error") is actually *sent or committed by the slave* (the remote, message-receiving station) and is only *recorded* by the master.

Following a successful message transfer (anticipated by a remote-echo count that equals the sent-message size), the software logs the number of the last remote-echoed-character reading session (here 3), the total number of message characters sent and the total number of remote-echoed chars retrieved (the last two numbers should be the same):

```
30.06.00 16:00:07 rem read 3 25000 chars sent, 25000 rem-echoed chars retrieved
```

As soon as the appropriate confirmation arrives from the remote station that it has stored the error-free message, we get what we've been waiting for: the statistics of the transfer:

```
30.06.00 16:00:10 Message "file.25k" filed.
LinkTime 2 s NegotiationTime 22 s
25006 bytes sent in 456 s; 54.8 bytes/s 35 TxARQs 1 TxERRs
30.06.00 16:00:10 BYE sent...
```

The link time is the time between the connection request and arrival of the ***CONNECTED string. This time is usually a second or two except in very poor conditions. The negotiation time is the time between establishment of the connection and the beginning of the message upload. This is the time taken up by sending, receiving and acknowledging the subject prompt, etc.

Next come the number of characters sent and the transfer time. The latter is the time between the start of the upload (right after the end of the negotiation phase) and confirmation from the message-receiver of correct message reception (here 456 seconds, or about seven-and-a-half minutes). The throughput in bytes per second (54.8 bytes/s) is the uncompressed message size (here 25006 bytes) divided by the transfer time.

The remaining statistics are the numbers of repeat requests and frame errors received by the master (message-transmitter). These are labeled TxARQs and TxERRs since they are recorded at the massage-transmitting station. Here there were 35 repeat requests and one error from the message recipient.

After logging the transfer data the program sends a disconnect request (“BYE sent”). Logging of a successful transfer is finished when the program receives the “***DISCONNECTED” prompt from the local modem:

```
30.06.00 16:00:21 Disconnected
Trial 1: Successful xfer to PTC-II MBOX
```

As noted above, we have found detailed transfer-log data such as this to be useful on many occasions in understanding protocol behavior and analyzing bugs in the testing software.

All of the files sent for this report consisted of readable text. Through experimentation we deduced that the “optimal” file size for throughput assessment of Pactor-II (close to highest throughput with smallest test time) was between 10 and 40k bytes, and most of our files had sizes in that range. This “optimal file-size range” for throughput assessment applies to most of the modern HF ARQ protocols we have evaluated.

4. Recorded Throughput-Data Format

The data archive file into which the results of each transfer were written by the test software contains the date-time group at the time (GMT) of the transfer, callsign of the receiving station, callsign of the sender, the mailbox system called (PTC=built-in Pactor-II or ArM=Airmail), the compression mode (0, 1 or 2), the frequency in MHz, the link time in seconds, the uncompressed message-file size in characters, the message transfer time in seconds, the throughput in characters/s, the negotiation time in seconds, the number of repeat requests and the number of bad frames from the file-receiver.

Here are excerpts from the NVIS transfer-data file for Pactor-II tests run in March and July 2000:

```
11.03.00 11:52:18 DER1   POR1  ArM 1    7.5xx  1 12010   306 39.2  32  12  7
11.03.00 12:02:10 DER1   POR1  ArM 1    7.5xx  2 20095   479 42.0  37  25  0
11.03.00 12:15:09 DER1   POR1  ArM 1    7.5xx  1 20093   430 46.7  23  30  0
11.03.00 14:12:01 BED1   POR1  PTC 2    3.1xx  1 20091   385 52.2  12   6  0
11.03.00 14:23:42 BED1   POR1  PTC 2    3.1xx  1 25006   460 54.4  12   3  0
20.07.00 02:45:49 KB1JY  KB1PZ ArM 0    3.615  1 20008   967 20.7  27  42  2
20.07.00 04:41:44 KB1JY  KB1PZ ArM 0    3.615  1 30010 1641 18.3  35 109  21
```

The first line, for example, records a 12010-character file sent at 11:52:18 GMT (6:52:18 AM local time) by POR1 in Portland, Me., to an Airmail mailbox at DER1 in Derry, N.H. The message was sent in compression mode 1. Linking took 1 second, negotiation 32 seconds and the transfer 306 seconds, for a throughput of 39.2 chars/s. There were 12 frame-repeat requests from Derry and 7 erroneous frames from Derry were received in Portland.

Files like this are opened and analyzed by a data-analysis program described in the next section.

5. The Data-analysis Software

The results in the data archive were analyzed off-line by a program called sum_ptc.c. This program reads the archive file line-by-line looking for various strings. As it moves through the file to the end-of-file indicator, the program keeps running totals of throughput and other data corresponding to the strings, from which it calculates statistics such as the average and standard deviation of the throughput. The statistics are written to a summary file after the pass through the archive file. Switches in the summary code are set before each run to pick out specific data (corresponding to various string combinations) for analysis. For example, we select lines with particular mode settings to pick out data sent in each compression mode, and use the date-time group to distinguish daytime from nighttime transfers. Since the summary program was written to analyze archive files of fixed format but arbitrary length, summaries of the data collected so far can be made at any time.

Shown below is the output of the summary program for all Pactor-II NVIS tests run from May 1999 to early February 2000 (a subset of all such data). For this output we set the software switches to compute throughput statistics for *files sent at night to a Pactor-II mailbox in compression mode 2.*

```
Statistics calculated from archive.ptc on 03.02.00 00:11:23
NIGHTTIME PTC-BBS COMPRESSED(2) PTC-II TRANSFERS
SAMPLE SIZE       = 56        E(SENT)            = 18730 bytes
E(TRANSFER TIME)  = 493  s    sd(TRANSFER TIME)  = 190   s
E(LINK TIME)      = 2    s    sd(LINK TIME)      = 1     s
E(NEGO TIME)      = 27   s    sd(NEGO TIME)      = 23    s    SAMPLE SIZE       = 48
E(THRUPUT)        = 40   cps  sd(THRUPUT)        = 17    cps  sd(mean_THRUPUT) = 2.3 cps
max(THRUPUT)      = 98   cps  Median(THRUPUT)    = 38    cps  E(TPUT/Hz)   = 0.080 cps/Hz
XFER FAILURES     = 1         P(XFER SUCCESS)    = 56/57 = 0.98
```

The output shows that the expected (average) throughput for 56 mode-2-compressed file transfers was about 40 characters per second (cps) and that the largest observed throughput in this mode in these conditions was 98 cps. The average sent-file size was 18,730 bytes. The sd(THRUPUT) reflects the spread of throughput measurements about their average. Roughly speaking, about two-thirds of a set of measurements will be within one standard deviation (here 17 cps) of their mean and over 90% will be within two standard deviations of their mean.

We also calculate the "standard deviation of the mean throughput" [sd(mean_THRUPUT)] in characters per second and the average throughput per Hertz of signaling bandwidth. The standard deviation of the mean throughput (equal to the standard deviation of the throughput divided by the square root of the sample size) is an assessment of the variability of the mean itself (which has its own statistical variability). The sd(mean) above suggests that our sample size in this case is big enough to make us confident that if we collected many more throughput measurements under roughly the same conditions, we would not get an average throughput that differed from the one above by more than about two characters per second.

To estimate the average throughput per Hertz [E(THRUPUT/Hz)], we divide the average throughput by the signaling bandwidth. For Pactor-II, the signaling bandwidth is 500 Hz. In this case the throughput per Hertz was 0.08 cps/Hz.

The average link, negotiation and transfer times were 2, 27 and 493 seconds. (We started measuring negotiation time a few days after the start of testing so the sample size of the average negotiation time is slightly smaller than the sample size of the throughput.)

The median throughput (the throughput above and below which half the samples lie) was 38 cps. The median can be used as a check on whether or not the mean is a good summary of performance. Since the median and mean are close, the mean is a realistic measure of performance in this case.

The last line of the statistical summary gives the percentage of successful file transfers. Unsuccessful transfers occur when, after a successful link, the number of times the modem tries to send a data frame exceeds a programmable limit (MAXERRS) causing the protocol to time out and terminate the link. (We set MAXERRS = 30.) As in our other testing, we do not include failures to link in our transfer success ratios. In the excerpt given above, 57 transfer attempts resulted in ARQ links, one of which was terminated when the link timed out before the message file got through. This led to a transfer success ratio of 56/57 or approximately 98%.

Figure 2 gives a scatter plot of all throughput measurements made to a Pactor-II mailbox through the beginning of January 2000 (only about half of the testing period). On the ordinate is throughput in characters/s and on the abscissa time-of-day (GMT). The plot gives a good picture of the spread of throughput, which is large, and typical of HF data communications. (Throughputs lie between about 10 and 110 cps.) One can discern in this plot the clear advantage of compression (Modes 1 and 2) over non-compression (Mode 0). Also discernable, perhaps, is the slight superiority of Mode-2 over Mode-1 compression.

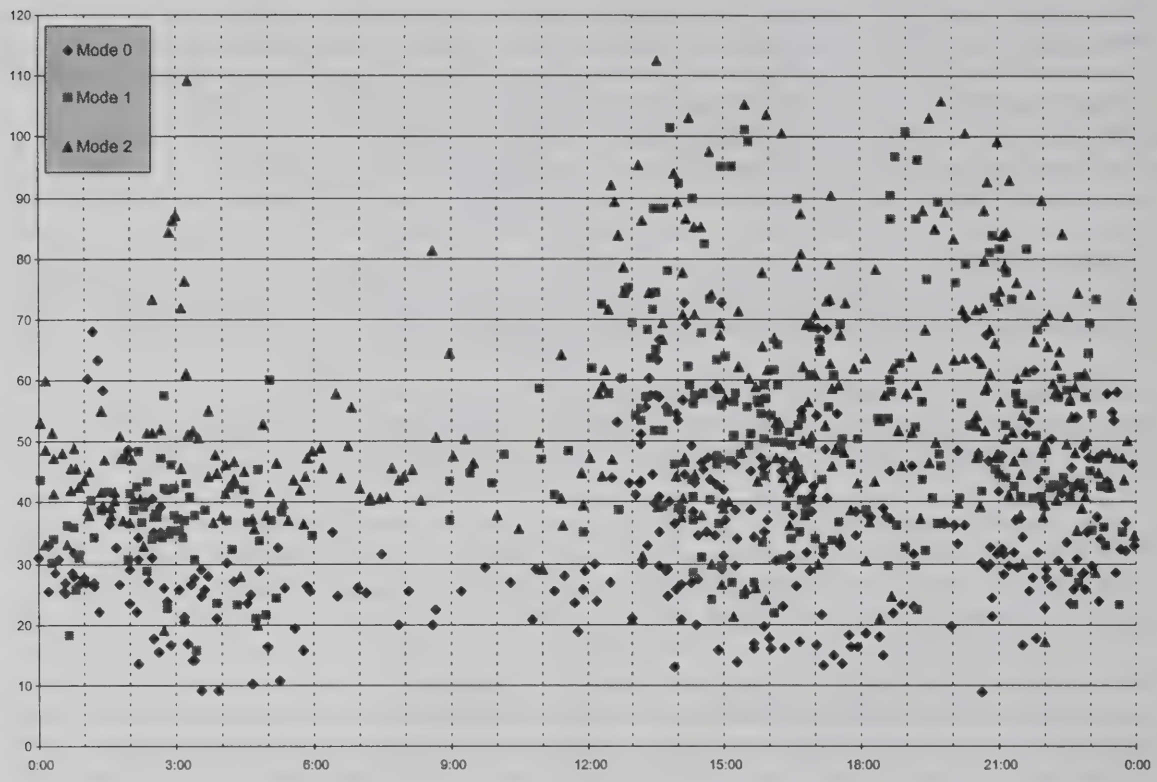


Fig. 2. Scatter Plot of Throughput to the Pactor-II Mailbox

6 . Statistical Summary of Throughput Results

The results of our NVIS tests of the Pactor-II system (as of July 2000) are summarized in Tables 1 through 6 below. They correspond to transfers to Pactor-II and Airmail mailboxes, in each case with compression modes 0, 1 or 2. The first column in each table gives the average throughput and its standard deviation, the average throughput per Hertz, the standard deviation of the mean throughput and the maximum observed throughput. The second column gives the number of transfers and the probability in percent of successful transfer [P(good xfer)] in each case. The third column gives the median throughput. The fourth column gives the mean of the link, negotiation and transfer times in seconds and the fifth column the average number of bytes in the original, uncompressed sent-message files.

Table 1. Pactor-II Mailbox Throughput Data in Compression Mode 0

File Type
&
Time
	E(thruput)
	sd(thruput)
	E(tput/Hz)
	sd_mn(tput)
	max_tput
		No. Xfers
		P(good xfer)
			Median
			Tput
				Average
				Link Time
				Neg. Time
				Xfer Time
					E(No_char)

Mode 0 Day	36 cps 15 cps 0.07 cps/Hz 0.9 cps 72 cps	288 97%	34 cps	2 s 18 s 608 s	18852
Mode 0 Night	34 cps 13 cps 0.07 cps/Hz 0.9 cps 87 cps	227 99%	30 cps	1 s 17 s 634 s	19963

Table 2. Pactor-II Mailbox Throughput Data in Compression Mode 1

File Type
&
Time
	E(thruput)
	sd(thruput)
	E(tput/Hz)
	sd_mn(tput)
	max_tput
		No. Xfers
		P(good xfer)
			Median
			Tput
				Average
				Link Time
				Neg. Time
				Xfer Time
					E(No_char)

Mode 1 Day	59 cps 24 cps 0.12 cps/Hz 1.5 cps 104 cps	245 99%	56 cps	1 s 16 s 518 s	23702
Mode 1 Night	45 cps 16 cps 0.09 cps/Hz 1.1 cps 102 cps	212 100%	42 cps	1 s 17 s 506 s	20735

Table 3. Pactor-II Mailbox Throughput Data in Compression Mode 2

File Type
&
Time
	E(thruput)
	sd(thruput)
	E(tput/Hz)
	sd_mn(tput)
	max_tput
		No. Xfers
		P(good xfer)
			Median
			Tput
				Average
				Link Time
				Neg. Time
				Xfer Time
					E(No_char)

Mode 2 Day	62 cps 23 cps 0.12 cps/Hz 1.2 cps 110 cps	359 99%	58 cps	1 s 16 s 445 s	23998
Mode 2 Night	47 cps 19 cps 0.09 cps/Hz 1.2 cps 109 cps	234 99.9%	44 cps	1 s 19 s 509 s	21522

Table 4. Airmail Mailbox Throughput Data in Compression Mode 0

File Type
&
Time
	E(thruput)
	sd(thruput)
	E(tput/Hz)
	sd_mn(tput)
	max_tput
		No. Xfers
		P(good xfer)
			Median
			Tput
				Average
				Link Time
				Neg. Time
				Xfer Time
					E(No_char)

Mode 0 Day	36 cps 14 cps 0.07 cps/Hz 1.0 cps 73 cps	210 97%	35 cps	1 s 23 s 626 s	19858
Mode 0 Night	30 cps 11 cps 0.06 cps/Hz 0.9 cps 68 cps	152 95%	28 cps	1 s 23 s 783 s	20480

Table 5. Airmail Mailbox Throughput Data in Compression Mode 1

File Type
&
Time
	E(thruput)
	sd(thruput)
	E(tput/Hz)
	sd_mn(tput)
	max_tput
		No. Xfers
		P(good xfer)
			Median
			Tput
				Average
				Link Time
				Neg. Time
				Xfer Time
					E(No_char)

Mode 1 Day	53 cps 19 cps 0.11 cps/Hz 1.1 cps 102 cps	270 98%	49 cps	1 s 23 s 474 s	22467
Mode 1 Night	41 cps 11 cps 0.08 cps/Hz 0.9 cps 74 cps	162 98%	39 cps	1 s 25 s 566 s	21502

Table 6. Airmail Mailbox Throughput Data in Compression Mode 2

File Type
&
Time
	E(thruput)
	sd(thruput)
	E(tput/Hz)
	sd_mn(tput)
	max_tput
		No. Xfers
		P(good xfer)
			Median
			Tput
				Average
				Link Time
				Neg. Time
				Xfer Time
					E(No_char)

Mode 2 Day	57 cps 20 cps 0.11 cps/Hz 1.3 cps 112 cps	244 98%	54 cps	1 s 22 s 446 s	22425
Mode 2 Night	48 cps 13 cps 0.10 cps/Hz 1.1 cps 109 cps	144 98%	46 cps	1 s 23 s 495 s	22622

7. Discussion of Pactor-II Performance Statistics

Tables 1 and 4 (transfers to Pactor-II and Airmail mailboxes in compression mode 0) show that the inherent (no compression used) over-the-air average throughput of Pactor-II on our NVIS links is around 36 characters per second (averaging the two mailbox cases). This is equivalent to an average delivery time of about ten minutes for a 20k text file. The nighttime NVIS throughput average for uncompressed files is about 32 cps. The standard deviations of these throughputs are around 15 cps in daytime and 20 cps at night. These imply that there is considerable variability in performance even with the same mode (see Fig. 2). This is typical of HF data transfers. The main cause of the lower nighttime throughput is probably QRM. Standard deviations of the means are about a character per second. This leads to high confidence that we have large enough sample sizes for our means to characterize real NVIS throughputs.

Figure 3 gives a plot of throughput vs. time of day for sixty-one 30k-files sent with Mode 2 compression half-an-hour apart. (The time between the end of one transfer and the beginning of the next was half-an-hour.) The transfers occurred over the course of 42 hours in early July 2000, when the ionosphere was relatively undisturbed. The link was from Derry, N.H., to an Airmail mailbox in Bedford, Mass. (30 miles away). The plot illustrates both diurnal and statistical variability of performance over NVIS channels. Note, in particular, the rapid rise in throughput at sunrise as the MUF increases, and the generally lower throughput at midday, which was caused by D-layer absorption.

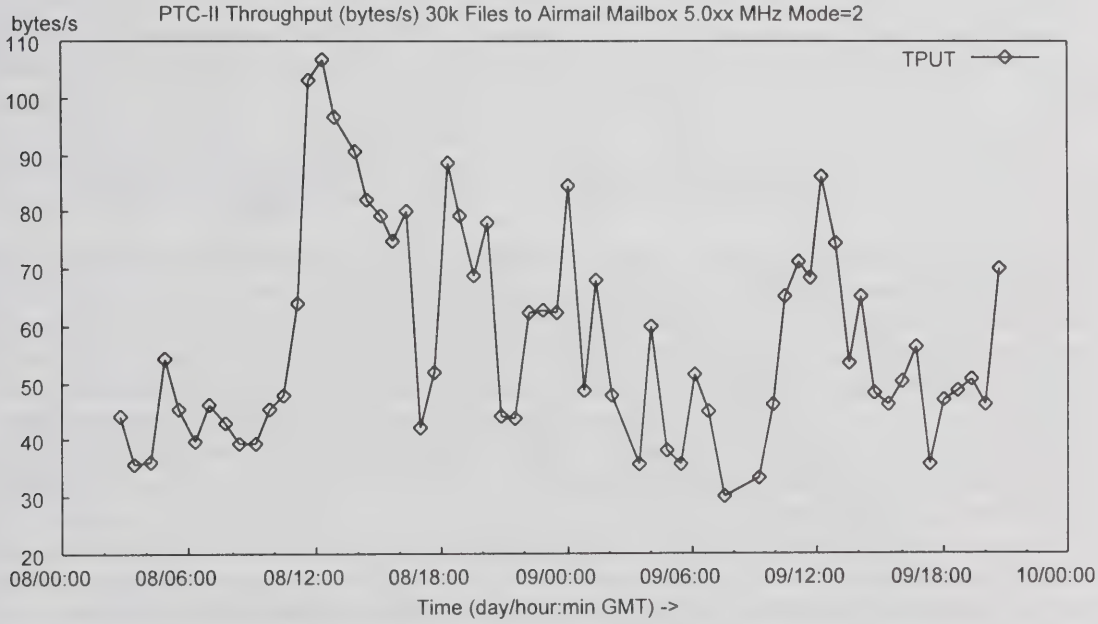


Figure 3. Throughputs for 61 Files Sent Half-an-Hour Apart

Measured maximum throughputs are generally large—at or near the theoretical maxima. These are not very useful statistics, however, since these maxima are rarely attained.

The probability of correct message reception was above 95% in all cases. This is also typical of well-designed HF ARQ protocols with their numbers of retries set conservatively. (Ours were generally set at 30.) This says that if Pactor-II can establish a link, it can almost always (even at night) complete a file transfer. (The transfer success rate given a link can be raised to almost 100% by raising the retry limit, at the expense of wear and tear on radios and amplifiers.)

Although we were usually successful in choosing frequencies that supported linking, a number of link attempts (perhaps 10% of them) failed when there was too much QRM on all available frequencies or when the MUF dropped below our set of available operating frequencies. An automatic link establishment (ALE) system that uses sounding data to choose frequencies for link attempts would probably have reduced this number if a large enough set of linking frequencies was available. Linking statistics are not included in our data: all of our transfer-failure percentages are conditioned on a link's having been established before each transfer attempt.

The Pactor-II and Airmail systems (the first in the modem, the second installed on Windows PCs) do not appear to have significantly different throughputs associated with them. For that reason we have combined their statistics in the following discussion.

Average throughputs for text files compressed in Mode 1 are about 55 cps during the day, and 43 cps at night (Tables 2 and 5). Mode 1 compression led to about 50% higher throughput than using no compression.

Mode 2 compression produced day- and nighttime throughputs of about 60 and 48 cps. This mode produced the best performance, probably owing to the wider choice of compression methods made by the protocol in that mode.

It should be noted that how much a compression method reduces the size of a file, and whether it even reduces it at all, depends on the file's type. Our results apply only to text files.

8. Throughput Comparison with Clover II and Certain Military Protocols

For some time newsgroup and listserver participants have been debating whether Pactor-II has higher throughput than HAL Corporation's Clover II system. Some of the claims in this debate seem to rest on anecdotal or otherwise questionable evidence. The debate is perhaps further clouded by the fact that HAL sells other Clover systems (the P38 and Clover2000) that generally have lower (P38) and higher (Clover2000) performance than Pactor-II. Since we have also carried out extensive on-air NVIS measurements of the Clover II system (as well as of the P38 and Clover2000 systems—see the references), results from our measurement campaigns may push the Clover II vs. Pactor-II debate a little closer to resolution.

Both Pactor-II and Clover II operate in 500-Hz bandwidths, use powerful forward error correction, have waveforms that are either PSK or fairly similar to it and employ sophisticated ARQ schemes that rapidly adapt modulation modes and FEC coding rates to channel conditions. Both systems have had their adaptation algorithms tweaked and tested over the air numerous times during several years with resultant firmware upgrades. These facts might cause one to be skeptical of claims that one system has much higher throughput than the other.

Before we compare the two systems' performance on NVIS links we should list the similarities and differences in conditions under which we tested each system.

Similarities

Links, transmitter output powers, antennas and average sent text-file sizes.

Differences

Transceivers, sunspot numbers, freedom to set "bias" for the Clover II protocol and scheduling of nighttime message transfers.

The differences deserve elaboration. Two of the four transceivers we used for some of the Pactor-II tests were Yaesu FT–1000MPs with 500-Hz SSB filters. These filters may have improved performance somewhat for the Pactor-II tests. However, since the frequencies we used for transfers to the Pactor-II and Airmail mailboxes were seldom bothered by QRM, this difference probably had little effect on results.

Our Clover II tests took place over six months between September 1997 and February 1998, when the sunspot number was about half what it was for the more recent Pactor-II tests. Although this didn't make much difference in the set of frequencies we chose for NVIS testing (mostly 3-5 MHz in both cases), there was more flare activity during the end of the Pactor-II testing period than during the Clover II testing. However, since scheduling conflicts kept us from testing during most flare aftermaths, sunspot number differences had little effect on our throughput data.

The Clover protocol allows users to set a parameter called *bias* that determines how aggressive the protocol should act in various perceived channel conditions. The biases, which are associated with frame lengths and the FEC coding rate, are Fast, Normal and Robust, with Fast appropriate when the channel is good and Robust when it's bad. We adjusted the bias during our Clover measurements. Although this added additional variance to our Clover results, comparisons of achieved throughput after most bias changes suggested that our changes usually raised or at least maintained throughput. We have concluded that Clover bias adjustments (viewed as an integral part of the protocol) did not distort our throughput comparison.

The differences in nighttime scheduling had to do with the ease of writing code to schedule tests with the Pactor-II and Clover II modems. Pactor-II has a relatively simple set of text-based commands and we were able to write testing programs for it relatively quickly. Clover II, on the other hand, has a rich and fairly complicated control language and we didn't have time to write and debug scheduling code with it. This forced us to run our nighttime Clover II tests in the *evening* (when we were awake) rather than spread them more evenly over the whole night. The effect of this is hard to assess accurately. On the one hand, rarely operating Clover II in the middle of the night avoided occasional operation above the MUF then, which may have lowered Pactor-II throughput on occasion. On the other hand, some NVIS frequencies are more crowded in the evening than in the middle of the night, which may have worked to the detriment of Clover II. We decided that these differences were murky enough to preclude a useful comparison of nighttime performance with our data.

A comparison of the daytime performance of the two protocols is given in Table 7. (Daytime was defined for both sets of data as 5 AM to 5 PM local time.) The table gives throughput and other statistics for compressed and uncompressed text file transfers. In the case of Pactor-II we chose Mode-2 data for comparison in the compression case, since Mode 2 produced the highest Pactor-II throughput. (Clover II uses a fixed, dictionary-based compression technique from the PKLib package.)

Our long-term NVIS measurements show Pactor-II with slightly higher daytime throughput than Clover II for text files sent in compressed or uncompressed modes. The differences in transceivers used and the fact that the tests were separated by more than a year are probably enough to explain the differences in throughput. The two systems had almost the same high probability of successful message delivery. Our data suggest that users wanting to choose between the Pactor-II and Clover II systems should use other criteria than throughput or message-delivery probability.

Table 7. Pactor-II vs. Clover II Performance

File Type & Time	E(thruput) Chars/s	Number Of Transfers	Maximum Thruput Chars/s	P(success) Percent	E(No_char)
Uncompr. Day					
Pactor-II*	36	288	72	97	18852
Clover II	34	239	69	98	31346
Compr. Day					
Pactor-II*	62	359	110	99	23998
Clover II	52	172	128	98	33986

* Pactor-II: Mode 2 to PTC2 MB, as of 30 June 2000

The results in Table 7 may be compared with those for the US Federal Standard 1052 and NATO STANAG 5066 protocols, which use a much wider bandwidth (usually 3 KHz) and much more expensive modems ($3k and up). Modern military modems usually have data-directed equalizers that deal very effectively with multipath. The currently fielded versions can operate at up to 2400 bits per second and adaptive military protocols often achieve that rate even on NVIS links.

Our measurements with the FS-1052 protocol implementation by the Harris Corporation have produced uncompressed average daytime throughput of about 125 cps for 20-to-30k text messages sent over a 30-mile NVIS link. This is almost four times what Pactor-II and Clover II achieve on average. (We are not aware of any large-scale on-air measurements made yet with the STANAG 5066 protocol, but we expect it to have throughput similar to that of FS-1052.)

9. Concluding Remarks

We hope that our data will aid understanding of Pactor-II file transfers over HF and perhaps serve as a useful introduction to how Pactor-II works. The system is now employed all over the world by amateurs (especially for sending E-mail to and from boats and recreational vehicles). A number of international aid and other communications-providing organizations also use Pactor-II to send information across parts of Africa, where alternative means of long-haul communication are not available, or too expensive. Our data may shed light on why this effective, amateur-developed modem and its associated protocols are so widely used.

Acknowledgments

We are grateful to Jim Corenman (KE6RK) and Rick Muething (KN6KB) for advice on Airmail and Winlink. Jim and Rick also gave us essential help understanding flow control and buffering in the Pactor-II modem.

References

1. Ken Wickwire (KB1JY) et al., "On-air Measurements of HF TOR and Packet Throughput, Part I: Near-Vertical-Incidence-Skywave Paths," *Digital Journal*, March 1996.

2. Ken Wickwire (KB1JY), "On-air Measurements of HF TOR and Packet Throughput, Part II: One-hop Skywave Paths," *QEX,* June 1996.

3. Ken Wickwire (KB1JY), "On-air Measurements of HF Data Throughput: Results and Reflections," *Proc. 15th ARRL/TAPR Digital Communications Conference*, ARRL, 1996.

4. Ken Wickwire, "On-air Measurements of MIL-STD-188-141A ALE Data Text Message Throughput over Short Links," MITRE Internal Report, 1996. (Preview in *Proc. 15th ARRL/TAPR Digital Communications Conference*, ARRL, 1996.)

5. Ken Wickwire (KB1JY) et al., "On-air Measurements of Clover P38 Throughput," *Proc. 16th ARRL/TAPR Digital Communications Conference*, ARRL, 1997.

6. Ken Wickwire (KB1JY) et al., "On-air Measurements of Clover II and Clover 2000 Throughput," *Proc. 17th ARRL/TAPR Digital Communications Conference*, ARRL, 1998.

Email: kwick@mitre.org. Packet radio: KB1JY@W1ON.

AUTOMATIC VOICE RELAY SYSTEM
AVRS

Bob Bruninga, WB4APR
115 Old Farm Ct
Glen Burnie, MD 21060

ABSTRACT: AVRS is a "VOICE" addition to APRS.

Just like APRS now provides mobile-to-mobile worldwide MESSAGE capability via the Internet APRServe system, there is no reason why we cannot add voice to this same type infrastructure for mobile-to-mobile voice. And AVRS is not limited to one channel. It can be expanded to as many channels as the locals want (ie, it can handle explosive growth)...

Internet connection of local Repeaters is not new. Iphone links have been going on for several years, but with the lack of any common user commandable access-on-demand and the competition for on-air time with conventional use of the repeaters, it remains as just a novelty. But, by tying this capability into the command structure and worldwide text messaging capability of APRS, the mobile user would then have total control and could "dial up" a link anywhere on the planet. Remember, he can already communicate by text message to confirm the other person is on the air. AVRS then lets him open a voice link to that same area.

INITIAL AVRS OPERATIONS:

For one channel (the first), simply connect a local VOICE SIMPLEX frequency in your area to the Internet with IPhone. A mobile who wants to chat to any other APRS mobile ANYWHERE, sends an APRS "AVRS link" MESSAGE from his mobile on 144.39 that ACTIVATES the AVRS link and while it is open, anything he says on that SIMPLEX channel goes via IPHONE to the other AVRS node anywhere.

Initially, until it became saturated, the link would probably just go to ALL, so that we would be able to find someone to talk to. The format for activating an AVRS link would be a short APRS message to the generic callsign of AVRS. Connections to other AVRS nodes could be by AREA or by SUBJECT matter or by CHANNEL number.

```
TO: AVRS   :Help         <= Responds with a list of local AVRS freqs
TO: AVRS   :ALL          <= would activate link to ALL unused AVRS sites
TO: AVRS   :Chicago      <= would activate link to chicago
TO: AVRS   :Annapolis    <= would activate link to Annapolis
TO: AVRS   :AMSAT        <= would activate link to all "AMSAT" channels
TO: AVRS   :ATV          <= would activate link to all "ATV" channels
TO: AVRS   :CH 1         <= would activate link to all "Channel 1's"
TO: AVRS   :Bye          <= would drop the link.
```

Thus, we have a worldwide VOICE channel(s), just like we have a PACKET channel. YES, as popularity grows, it will be a ZOO!, but then you simply expand the number of RADIO channels to separate out the users by SUBJECT, or by CHANNEL NUMBER. What we end up with is just like the growth potential of CELLULAR. Each local area can use ANY simplex channels they want and can MAP any "channel designation" to any channels. This is ALL a local issue. For example, Chicago might have 3 channels on the air serving for chats on the subjects of APRS, AMSAT and RELIGION. But In Annapolis where there is little or no activity at any instant, all three of these "subject" channels would be linked to our one hardware channel which would relay which ever one was currently in use. In such a case, only the first connection works, since Iphone can only support one connection at a time.

Notice how the above "HELP" message allows you to find out about AVRS in the local area where you are. If there is an AVRS in the area, it will respond with the list of channels and frequencies to tune into in your local area. Responses from the local AVRS system will also be by packet such as:

```
TO: yourcall :AVRS Greenbelt is 28.910, AMSAT is 28.916
TO: yourcall :Connected to Chicago on 28.913
TO: yourcall :Annapolis is busy
TO: yourcall :AVRS not available
TO: yourcall :Connected to AMSAT on 28.916
TO: yourcall :Goodbye. AMSAT link on 28.916 is deactivated
```

AVRS CHANNEL FREQUENCY BAND:

So that you can do ALL this from your mobile, without interference to APRS on 144.39 and other mobile activities on 2 meters and 70 cm, we think that 10m SSB is the ideal band/mode for these simplex AVRS voice channels for several reasons:

1) NEW band so AVRS does not QRM your other mobile APRS and 2m activity
2) NEW band so AVRS nodes can be collocated at existing HIGH node sites
3) Simplex range to mobile is 30% greater than on 2m or 440?
4) Band has many many SSB channels available
5) Avoids political battles on 2m, 70cm
6) New radios cost only $149, Antennas are FREE (CB magmounts)

The one disadvantage is band openings. But hey, if the BAND IS OPEN, then turn this AVRS kludge OFF and talk DIRECT! Band openings are only a problem during a few hours of a day and only during a few years out of every 11. By the time we get this going, we will be out of the solar maximum and so we will see the next 8 years of DEAD BAND on 10 meters. Also, since you have the 10m rig in your car, you will HEAR that the band is open before you open a link... Think of AVRS on 10m as only a crutch, like repeaters, for use only when the band is dead.

10 METER CHANNEL RANGE:

Although the primary advantages of 10m are the ability to add the band to existing mobiles and Igates without interference to priority use of the other APRS and mobile bands, there is supposedly an additional advantage of extended simplex range on 10m, although so far, some practical experience has not shown the same conclusions.

From the ARRL FM & REPEATER MANUAL, using graph 6-18 in the 1970 edition, yields the following approximate (eyeball) results. This analysis takes into account not only the better link margin with longer wavelength but then also the better antenna gains available with the shorter wavelengths. One thing the book was not clear on, was the gain of the BASE antenna for the graph. I took the conservative approach and assumed that they also were 1/4 wave radiators, thus I applied better antenna gains for both the mobile and base as shown in the 6th column below:

	FOR 100' HAAT 1W		FOR 25W	FOR 40'	FOR 40'	RESULT	SSB
BAND	EQUAL AREA ANTENNAS	QUARTER WAVE WHIPS	QUARTER WAVE WHIPS	QUARTER WAVE WHIPS	ACTUAL MOBILE-BASE ANTS	ACTUAL MOBILE RANGE	SAME ANTS
70cm	18	8	13	11	5/8 dB	20 mi	28mi
2m	22	14	23	20	3/6 dB	27 mi	34mi
10m	25	28	42	32	-3/3 dB	36 mi	43mi

The RESULT column assumes high gain antennas at the base and typical mobile antennas for the bands involved. Note, that I assumed - 3dBd for the mobile 10m antenna assuming everyone would use a shortened loaded whip magmount. The final column assumes a 7 dB advantage of SSB over FM based on the Bandwidth ratio.

Still, though, no one has given practical experience that supports these numbers, and in fact, many suggest that 10m is no better, if not worse than 2m in general. But range is not the primary factor in the AVRS application, the new-band-non-interference-with-existing-mobile-ops and low-cost of the 10m SSB radio system are the primary advantages of 10m. As a sampling, here are some preliminary opinions received from others on the APRSSIG:

Walter Holmes wrote:
> We have a 50W 10 meter FM remote base... at about 150 feet. The range is
> about 25 to 30 miles to a 25 watt mobile... about the same as 2m...

WO4U said:
> We experimented [on] 2 M and 10 M ... using a multi element beam the two
> frequencies seem to behave similar ... but 2 M (whips) always wins for
> distance per watt when going beyond about 5 - 7 miles. [But the]
> tests were all conducted in East TN - which is very mountanous...

MY SIMPLE TEST:
> I listened to the local APRS 10 meter feed while driving home from the site.
> The feed antenna was in clear with 80' above local terrain out 2mi. But
> Local terrain is at sea level. Actual HAAT is probably much less than 20'
> if not below zero in the 15 miles I drove home. At 6mi and beyond it was
> nominal weak signal good copy with no AGC action. Only minor additional
> reduction out the full 15 miles to home.

Since most 10m activity is for the DX when the band is open, my disussions with other operators reveals very little experience with it as a Base-to-mobile channel, so this paper and this subject is ripe for further experimentation.

CONCLUSION:

For $149 and a CB magmount, AVRS should be a POPULAR capability that will REVOLUTIONIZE amateur mobile voice connectivity just like APRS revolutionized mobile packet. Initially, if operators disagree with the choice of the 10m band as the ideal AVRS band, they can even put it on 2m or any other band from their shack to see if the concept will fly. It does not matter, it is purely a local issue.

We have everything we need, off the shelf, including IPhone and APRS and the radios... All we need is the interest "and a little software" to glue it all together… And the hope that Radio Shack will continue to sell the low cost radio even after the solar max…

de WB4APR, Bob

Notes

Published by: ARRL The national association for AMATEUR RADIO